2002

Extending a quantifier scope resolution algorithm by accounting for negation.

Sudhakar Reddy. Pareddy
University of Windsor

Extending a Quantifier Scope Resolution Algorithm by Accounting for Negation

by

Sudhakar R. Pareddy

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in Partial
Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science at the
University of Windsor

Windsor, Ontario, Canada
2002

973194

Abstract

This thesis report investigates a central problem to natural language understanding, namely the problem of scope ambiguity. The types of scope ambiguities that are considered are those that are generally resolved by speakers of a given language by relaying on common knowledge. Typical of this is the problem of resolving quantifier scope ambiguity. In this report we investigate the thesis that the QC algorithm of Saba and Corriveau can be extended to account for negation. We propose an extension to Saba and Corriveau QC algorithm.

To My Parents Ankamma & Pareddy Rami Reddy

"To make our computers easier to use, we make them more sensitive to our needs. That is, make them understand what we mean when we try to tell them what we want. If we want our computers to understand us, we'll need to equip them with adequate knowledge. Only then can they become truly concerned with our human affairs"

Marvin Minsky, 2000.

Acknowledgements

I am indebted to my teacher Dr. Walid S. Saba for introducing me to the subject for guiding me to approach problem solving. I am also indebted to my mother who always prays for the education of her children and encourages us to study though she has studied only up to class 6^{th}. I thank Dr. Walid S. Saba for providing intellectual support, which I rarely received from any one in past. I also thank the members of LEG group for the conversations we had and the insight they provided during those contestations. I thank my father Rami Reddy Pareddy, my brother Raghava Pareddy and my fiancée Prashanthi Karnati for their support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Natural Language Understanding

Since the invention of a machine that calculates simple mathematical formulas there was a vision that it can perform various functions that are needed by us. One such function is a need to communicate with humans that gave roots to natural language understanding (NLU), which is an active area of research in artificial intelligence (AI)[1].

Turing thought that natural language communication with machines is the ultimate test for intelligence (the Turing Test). Having an adequate natural language communication system cannot be underestimated. Over the past decades NLU research has developed not only various approaches to dealing with certain linguistic phenomena, but also various approaches to representing this complex system of communication. In 1950's the problem was overly underestimated, and various attempts to build machine-translation systems failed miserably. In the 1970's and 1980's many AI-based approaches also failed: either because they were based for the most part on ad hoc AI procedures that did not scale up, or they were highly formal and did not deal with commonsense reasoning and other pragmatic and psychological phenomena. The new trend, however, is to formalize commonsense reasoning by developing formal systems that model various aspects of human non-monotonic reasoning, such as temporal reasoning, default reasoning, fuzzy and uncertain reasoning, abductive (analogical) reasoning, etc.

McCarthy (2001) in his answer to the question "how is AI research done?", said "based on studying and formalizing common sense facts about the world and the problems that the world presents to the achievement of goals". In McCarthy (1990), he says "one path to human-level AI uses mathematical logic to formalize common sense

[1] While not exactly the same, in this report we will use the terms natural language understanding (NLU), natural language processing (NLP) and computational linguistics (CL), somewhat interchangeably.

knowledge in such a way that commonsense problems can be solved by logical reasoning. If a commonsense reasoning problem is well presented, one is well on the way to formalizing it".

Ambiguity in natural language is a ubiquitous problem. Some of these ambiguities can be resolved using commonsense knowledge like lexical disambiguation, prepositional phrase attachment, reference resolution, quantifier scope ambiguity, etc. Our main concern here will be the problem of quantifier scope ambiguities.

1.2 The problem of 'Quantifier Scope Ambiguity'

In English, quantification is expressed by a determiner combined with a common noun (or a more complex nominal) to form a noun phrase. In natural language the same effect is achieved using First Order Predicate Logic (FOPL): by variables, which play the same role as names and quantification rules, to form an expression. A setback in FOPL is that (i) a sentence may contain free occurrences of variables (i.e., how do we represent the meaning of a free variable in isolation?) (ii) the sentences with same type of expression may have infinitely many distinct syntactic analysis which turn out to have same semantic interpretation. Montague's Proper Treatment of Quantifiers over comes this draw back. That is Montague formulated a method by which noun phrases (i.e., proper noun or a determiner followed by a common noun) denotes objects of the same type, and consequently, this required a method by which one could assign meanings to partial expressions (i.e., expressions obtained for a noun phrase: a determiner combined with a common noun). We cannot assign semantic values to the partial expression at that syntactic level. Montague resolved this problem by utilizing the tool of lambda calculus to generalize FOPL expressions.

A sentence may involve more than one quantifier. Quantifiers in natural language can take scope wider than where they occur explicitly. Ambiguity of a sentence containing multiple quantifiers is accounted by assigning different logical forms, which

differ in the scope assigned to these quantifiers. This is termed as 'Quantifier Scope Ambiguity'. This ambiguity is accounted by the processes such as quantifier rising or quantifying in at the level of logical form. Each of these derived logical forms may differ in the meaning implied by it. A number of scope resolution algorithms have been developed by computational linguists some of them generate the possible readings by disallowing some readings due to syntactic or semantic criteria and others that rank the possible readings by order of preference.

Syntax and semantics alone are not sufficient in the resolution of quantifier scope ambiguity. Most computational linguists have mentioned that pragmatic information is needed to obtain the correct reading. Kurtzman & MacDonald (1993) mention that real world knowledge is needed to determine the degree of plausibility of one scope reading over the other. It is not yet clear as at which level to address resolution of quantifier scope ambiguity (i.e., at syntactic, semantic or pragmatic level). It is assumed that pragmatic is the last resort in case syntax and semantic fail to address.

1.3 The Problem Addressed in this Report

1.3.1 The Problem of Negation

The work described in this thesis report investigates an approach to resolving quantifier scope ambiguity by Saba & Corriveau (1999), and in particular approaches to how negation can be handled in the QC algorithm.

Computational linguists extensively studied the problem of resolving quantifier scope ambiguity in natural language. For example, (refe1, 1999; ref2, 1999) developed a number of scope resolution algorithms that essentially generate all possible scope readings, and incrementally discard readings that are not consistent with some syntactic or semantic criterion, and possibly rank these readings by order of preference. There are a number of problems with this approach that we discuss in detail below.

In recent years a solution to the resolution of quantifier scope ambiguities at the pragmatic level has been suggested by (Saba, 1999; Saba & Corriveau, 2001). In their approach a quantificational constraint (QC) that reflects an individual's belief as to how a certain relation can be manifested in the real world is used to derive a set of constraints on the plausibility of a logical form. Using these logical constraints (that model some pragmatic, or commonsense constraints), logical entailment can be used to reject scope orderings that are inconsistent with some pragmatic considerations, thus determining the plausibility of a scope reading. However, this process gives us a yes or no answer regarding the plausibility of a scope reading, and this is not sufficient when there are more than two *valid* scope orderings. The process of logical entailment is extended to a numerical algorithm (QC algorithm) in order to determine the ***degree of plausibility*** of a scope reading thus accounting for the most plausible scope reading.

The QC algorithm (numerical algorithm) however does not directly account for negation, although linguistically, negation functions as any other generalized quantifier. Note, however, that unlike other generalized quantifiers (such as *every, some, more than six, all but one, at least three,* etc.) negation is partly a linguistic and partly a logical phenomenon. In fact, at the level of logical form, negation always results in the unary logical 'not'. To illustrate, consider the meaning of *no* (Montague, 1973):

$$[\text{no}] = \lambda P \lambda Q[\forall x[P(x) \rightarrow \neg Q(x)]]$$

That is, '*no P Q*' is true if whenever something satisfies *P* it fails to satisfy *Q*. Thus at the level of logical form (LF), 'no' translates to universal quantification and *logical* negation. The question that arises now is how does a scope resolution algorithm take this into account, and at what level should negation be treated.

The scope resolution algorithm that we intend to extend is a numerical algorithm that eventually computes the relative ratios of quantified sets. The problem of negation, however, is that the relative ratio approach, which works well in estimating the degree of plausibility of a particualr scope ordering given an individual quantification constraint, would not work properly with negation. The point here is that while the relative ratio of (Many X) to (Most Y) is a meaningful number (assuming the cardinality of X and Y are known), the relative ratio of (No X) to (Q Y), for any quantifier Q, is meaningless, since the range of quantification in (No X) is an empty set, regardless of the cardinality of X. The problem is even more severe if we were to consider the relative ratio of (Q X) to (No Y) as we would have to consider division by 0 and taking limits.

A numerical treatment of quantification, along the lines of the QC algorithm, thus leads to a problem when dealing with negation. Fortunately, however, 'no' is only a *linguistic* determiner, not a *logical* quantifier. That is, logically, ((No X) Y) is equivalent to ((FORALL X) not Y), while 'not' here is the unary *logical* operator. This in fact presents an opportunity to extend the QC algorithm by accounting for negation, while keeping the advantages of the QC algorithm as will be discussed in some detail below.

1.3.2 The Thesis

The thesis we defend in this report is the following:

The QC algorithm can be readily extended to account for negation.

The motivations for extending QC algorithm to account for negation are the following:

1. There is a need for sound and formal investigation of the role the negation words play in combination with other quantifiers in a sentence with multiple quantifiers.
2. Maintaining the advantages of a formalized commonsense reasoning approach to resolution of quantifier scope ambiguity.

We propose an extension to the QC algorithm, of formalized commonsense reasoning strategy to the resolution of quantifier scope ambiguity that combines logical reasoning and commonsense data, to handle negation. It is easy to see that by extending the QC algorithm to account for negation word 'No', other negation words like 'Not all', etc., can also be handled with relative ease.

1.4 Outline of the Proposed Solution

The solution described in this report extends the QC algorithm to account for negation. Negation words like 'No' (or 'Not-all', etc.) are treated at the logical level, and not the linguistic level. Thus, a word like 'no' is given the meaning $\lambda P \lambda Q [(\forall x)(P(x) \rightarrow \neg Q(x))]$ thus pushing negation on to the verb and 'every' taking the position where 'No' occurs in the surface structure. At this point the QC algorithm can work, as before, although now one needs to adjust the predictions of the QC algorithm to obtain a set of rules concerning the role of negation. I considered all the various combinations of 'no' with other quantifiers (No C1 R Q1 C2), (Q2 C1 R No C2) and (Q1 C1 did not R Q2 C2), where Q1 and Q2 are quantifiers (other than 'no') and where R is a binary relation between two quantified concepts C1 and C2. I pushed surface structure negation onto the relation (as a unary logical operator). I then applied the QC algorithm and adjusted the algorithm. If negation can be handled by the QC algorithm, then there should be a general set of rules that could be added to the QC algorithm that reflect the role negation plays in quantifier scope. As anticipated, this was

in fact the case, which, incidentally, seem to strengthen the view of quantification advocated by the QC algorithm.

The work is original in the following respects:

1. The extension to QC algorithm accounts for negation word 'No' in a sentence with multiple quantifiers.
2. Analysis of this solution shows that other negation words like 'Not all', etc., can also be handled with relative ease.

I incorporate changes in the existing Natural Language interpreter for quantifier scope ambiguity resolution to handle negation.

1.5 Structure of the Thesis

The remainder of the thesis is structured as follows: In chapter 2, a review of related work is given. In chapter 3, an extension to the QC algorithm that accounts for negation is described. In chapter 4, implementation details of a Natural Language interpreter for quantifier scope ambiguity are discussed. In chapter 5, implementation details for incorporating negation in the existing Natural language interpreter for quantifier scope ambiguity are discussed. The changes incorporated allow obtaining the meaning of a sentence involving negation. In chapter 6, the conclusions drawn from the experiment are discussed and in chapter 7 some future work is suggested.

Chapter 2

Logical Form

Giving *a semantics* to a language is to provide an adequate map between *the syntax* of a language and some internal representation that captures all (and only) the intended meaning(s) of a given sentence in that language. That internal (and intermediate) representation of the meaning implied by some surface structure is termed Logical Form (LF)[2], and it is to sentences at this level semantic clauses are applicable (May, 1989). In defining the notion of semantics and meaning, Allen (1995) describes Logical Form as representation of context-independent meaning; he describes semantic interpretation as the process of mapping a sentence to its logical form.

Typically a sentence will have numerous possible syntactic structures, each of which might have numerous possible logical forms; moreover sentences may have multiple meanings due only to lexical ambiguities. Number of resolution algorithms are developed for all types of ambiguities, such as algorithms for lexical disambiguation, anaphora resolution, resolution of quantifier scope ambiguities, prepositional phrase attachments, etc.

2.1 Problems in Logical Form

There are many kinds of ambiguities in natural language. One particular type of ambiguity arises in the representation of natural language expressions due to the choice between various possible scope orderings between several operators (scope ambiguity), or between various possible scope orderings that are caused due to the interaction of

[2] The term 'logical form' is unfortunate as it implies mathematical logic is the only tool that has been suggested to capture meanings. In fact, semantic networks, derivation (or abstract syntax) trees, graphs, etc. can all be (and have been) used to represent meanings.

quantifiers in the sentence (quantifier scope ambiguity), the choice of the meaning to be selected for a sentence (prepositional phrase attachment), the choice of what the pronoun refers to (referential ambiguity), etc. Various ambiguities in natural language and the problems faced in the logical form are discussed in apendix A. The common theme among the examples presented in the appendix is that the resolution of most types of ambiguities seems to require some inferencing procedure that utilizes commonsense knowledge.

Quantifier scope ambiguity (QSA) in natural language has occupied the minds of linguists and logicians for decades. This ambiguity is due to the choice between various possible scope orderings of a sentence with multiple quantifiers. Deciding on a single and efficient logical form as a representation for a sentence with multiple quantifiers would be a major achievement for both linguists and logicians. The challenge is to arrive at logical form that captures all possible scope ordering, but one that is not too general so as to admit incorrect inferences. For example, one would like the LF corresponding to

Most graduate students attended a seminar on AI

to admit the two interpretations where one specific seminar was attended by most graduate students, as well as the reading implying the (more plausible!) meaning that most graduate students attended some (not necessarily the same) seminar on AI. Having one LF representing both of these interpretations is a very challenging task, because one has to be careful to not produce an LF that would allow incorrect inferences. For example, '*Most graduate students attended a seminar on AI*' should entail '*Most graduate students attended a seminar*', also it should entail '*Most students attended a seminar on AI*'[3].

[3] The relevance of this to an NLU system is as follows: if a knowledge base (KB) was told '*Most graduate students attended a seminar on AI*', then to respond 'appropriately' to the queries '*Did most students*

Linguists and logicians have introduced techniques such as quantifying-in and quantifier raising respectively. Computational linguists developed quantifier scope resolution algorithms which obtained the possible scope readings by discarding those scope readings that do not meet certain syntactic or semantic criteria and In some algorithms these readings might be ordered by preference. An approach to resolving quantifier scope ambiguity by using commonsense knowledge has also been developed. This approach obtains the plausible scope readings by discarding those scope readings that do not meet certain pragmatic criteria and determines the degree of plausibility of each scope reading.

In this chapter I review these approaches to resolution of QSA and further divide the chapter into four sections, in section 2 I discuss the specific problem of QSA, In section 3 I discuss various approaches to QSA, In section 4 I discuss the QC algorithm and In section 5 I discuss the problem of negation.

2.2 The Problem of Quantifier Scope Ambiguity

Quantifier scope ambiguity arises when two or more noun phrases (NPs) in a sentence contain a quantifier term such as every, some, a, many, several or few in determiner position. A sentence with n quantifiers has potentially n! distinct interpretations that are due only to the different possible scope orderings of quantifiers. For example (1) can admit 2! Interpretations given in (1a) and (1b):

(1) Every newspaper advertised a commodity.

(1a) $\forall n(\text{Newspaper}(n) \rightarrow \exists c\,(\text{Commodity}(c) \wedge \text{Advertised}(n,c)))$

---

attended a seminar on AI?' and '*Did most graduate students attended a seminar*' the KB system must make valid inferences, and thus the representation of the sentence must be such that it captures all (but only those) intended inferences. For more on this see (Saba and Corrivea, 2001) and Hobbs (1996).

[Read as: "For every newspaper n, there is a commodity c, such that n advertised c"]

(1b) $\exists c(\text{Commodity}(c) \wedge \forall n(\text{Newspaper}(n) \rightarrow \text{Advertised by}(c, n)))$

[Read as: "There is a commodity c, such that for every newspaper n, n advertised c"]

In a truth-conditional model of interpretation (Montague, 1974; Cresswell, 1973), where the interpretation of a sentence is the set of conditions under which the sentence is true, deciding on the correct interpretation of a sentence like (1) is crucial because the possible scope readings may not always describe the same state of affairs but potentially describe two distinct states of affairs.

That is, although scope readings (1a) and (1b) derived from 1 differ only in the relative scope orderings of their quantified terms they may possibly describe different situations. According to the interpretation in (1a) for every newspaper there is some commodity that the newspaper advertised, and not necessarily, one commodity per newspaper. According to the interpretation in (1b) there is one particular commodity that all the newspapers advertised.

Linguists and Logicians extensively studied the quantifier scope ambiguity. They developed processes to explain how a seemingly unambiguous sentence can admit different interpretations that are due only to different quantifier scope orderings. Linguists developed a process called quantifier raising, which gives syntax level explanation of scope ambiguities. Logicians developed a process called quantifying-in (Montague, 1974). Quantifier rising and quantifying-in are syntactic level and semantic level explanation of quantifier scope ambiguity respectively. That is quantifier scope ambiguity is explained by obtaining a different derivation tree for every possible scope ordering.

Computational linguists (Hobbs and Shieber, 1987; Allen, 1987; Moran, 1988; Moran and Pereira, 1992; Park, 1995) developed scope resolution algorithms, which obtain the possible scope orderings and ranked these scope readings by order of

preference by use of some syntactic rules. These syntactic rules were assumed to be universal but this is not the case and the experiments in (Kurtzman and MacDonald, 1993) suggest that structural principals alone do not always predict the preferred scope ordering. Other principals such as lexical bias and world knowledge also seem to be operative in some situations.

2.3 Various Approaches to Qauntifier Scope Ambiguity

Since the late 1980's computational linguists have developed number of scope resolution algorithms (Hobbs and Shieber, 1987; Allen, 1987; Moran, 1988; Moran and Pereira, 1992; Park, 1995). Typically, a scope-neutral (Allen, 1987), or a quasi-logical form (QLF) (Alshawi, 1990) is constructed. A naïve algorithm for generating quantifier scopings is to generate all the permutations of the quantifiers. For a sentence with n quantifiers these algorithms will generate n! possible scope orderings that must be considered. A number of rules are then applied to reduce the number of permutations considerably to generate only the most likely or valid reading.

These approaches face a number of problems. The following are some of them:

- They suffer from the combinatorial explosion problem.
- They do not explain the transformation puzzle or the problem of branching quantifiers.
- These rules are assumed to be "universal" (i.e. they apply equally to all individuals; although there is evidence that different individuals have different scope preferences (kurtzman & MacDonald, 1992).
- The syntactically motivated preference rules do not always work.
- They do not help in deciding the most plausible or preferred reading at the end.

Below I discuss these problems in brief.

2.3.1 A Combinatorial Puzzle

The sentence in (2) has 5 quantifiers thus giving rise to 5! = 120 scope orderings. However, according to Hobbs and Shieber (1987) this sentence has 42 valid readings as opposed to 120.

(2) Some salesman of every department in most companies saw a few samples of each product.

As noted by Poesio (1996), a sentence such as (3) would have "hundreads of thousands of scopically distinct readings if all the permutations of scope-taking elements were considered admissible readings. Yet, human beings appear able to deal with these sentences effortlessly".

(3) A politician can fool most voters on most issues most of the time, but no politician can fool all voters on every single issue all of the time.

As argued (from varying perspectives) by (Cherniak, 1992; Johnson-Laird, 1994; Corriveau, 1995; Saba, 1995) a combinatorial approach where all syntactically valid permutations are incrementally considered is neither cognitively not computationally plausible. The point here is that unlike the idealized systems of logic, a computational system performing linguistic inferences must take into account a number of psychologically tested constraints, such as time constraints and limitations on working memory (see Corriveau, 1995).

Human agents draw a variety of inferences effortlessly, spontaneously, and with remarkable efficiency. This remarkable human ability seems paradoxical given the results about the complexity of reasoning reported by researchers in artificial intelligence. Given the speed by which humans comprehend large linguistic fragments, making a number of inferences along the way (Shastri & Ajjanagadde, 1993), it is highly unlikely that readers initially consider all the syntactically and semantically valid interpretations.

2.3.2 Transformation Puzzle

Perhaps one of the strongest indications of the pragmatic and inferential aspect of quantifier scope, according to (Saba & Corriveau, 1997), is the so-called "transformation puzzle". Many syntactic theories in the Chomskian tradition suggest that the active and passive forms are semantically equivalent. However, a reader of the following sentences can easily verify that the meanings implied by the two forms gradually become radically different.

(4) (a) Every artist drew a painting.
(b) a painting is drawn by every artist.

(5) (a) Every artist attended a conference.
(b) a conference is attended by every artist.

(6) (a) Every artist developed a skill.
(b) a skill is developed by every artist.

In (4) the meaning implied by the active and passive forms are quiet similar. (i.e., no matter how we read the sentence whether in active or in passive form only one reading comes to our mind that is "every artist drew some/a different painting"). In (5) however the situation seems to gradually change as, both the forms implying one or several conferences can be attended. In (6) the meanings implied by the active and passive forms are quiet different. (6a) Means that every artist developed some/different skill, while in (6b) there seems to be strong statement being made that every artist developed the same skill.

This 'gradual' change in the salient readings seems to suggest that there is some scalar aspect to whatever parameters this process is a function of (Saba & Corriveau, 1999). Lacking any syntactic or semantic explanations these examples suggest that some process that lies beyond syntax and semantics must be operative (Saba, 1999; Saba & Corriveau, 2001).

2.3.3 Beyond Syntax and Semantics

We have seen in section 2.2 that a sentence involving quantifiers has a number of readings due only to the scope orderings of quantifiers. In the introduction to section 2.3 we have seen that a number of approaches have been proposed to obtain the possible scope orderings. Here my argument is to see if the preferred of different scope reading of a sentence involving multiple quantifiers can be captured by the semantics or is common sense knowledge needed to attain this. As we go through a number of examples below we see that at each stage the argument that the selected scope preference of a sentence by the reader is captured by semantics by attributing the difference in selection to the lexical item in the head noun, main verb, etc., fails.

Consider the following example:

(9.1) / (9.2) Every { consulate member / ambassador } of an American embassy is summoned.

Each of the above two sentences has two possible readings that are due to the scope orderings of the quantifier terms. A reader of the above sentences might think of a wide scope 'a' in (9.1) (i.e., of the same American embassy... every consulate member...), but are not likely to think of a wide scope 'a' in (9.2) (i.e., of the same American embassy... every ambassador...). Though the above two sentences have the same form but only differ in the lexical item of the head noun, a reader selects different

scope preferences for these sentences. What common knowledge says (i.e., that embassies have many members but have one ambassador) does not concern semantics. This common knowledge is part of the reason why different scope preferences are selected for these sentences (Saba & Corriveau, 1999).

Consider now the following examples:

(10)
(11) Washingto n post Journalist $\left\{\begin{array}{c}\text{visited}\\ \text{publicized}\end{array}\right\}$ a company in every country.

The possible readings obtained by quantifier-rising or quantifying-in are equally valid, as far as semantics goes. The reader of the above sentences will select a different scope preference for each sentence, that is a reader will not think of a wide scope 'a' (i.e., physical existence of the same company in every country) in (10), but will admit a wide scope 'a' (i.e., same company) in (11). The only difference in (10) and (11) seems to be the main verb so we can argue that the difference can be contributed to the semantics of the main verb, as has in fact been suggested by a number of authors (e.g., Scha, 1981;Van der Does, 1994). Is this always the case? To explain this further consider another lexical variation of (10) and (11):

(10)
(12) Washington post Journalist visited a company in every $\left\{\begin{array}{c}\text{country}\\ \text{year}\end{array}\right\}$

We see that the sentences (10) and (12) differ only in the noun of the prepositional phrase but the choice of the preferred scope ordering selected by the reader is different, in that the physical existence of the same company in every city can not be

thought about in (10) where as in (12) the person can be understood to visit the same company in every year.

In examples (9.1) and (9.2) the selection of the preferred scope ordering can be attributed to the difference in the head noun, where as in examples (10) and (11) the selection can be attributed to the difference in the main verb but in examples (10) and (12) the selection can be attributed to the difference in the noun of the prepositional phrase. The differences in these sentences reveal a fact that the selection of a preferred scope reading of a sentence involving quantifiers is a function of a number of parameters. Thus, an appropriate scope resolution algorithm would have to be a function of (the meaning of) various constituents (Saba & Corriveau, 1999).

Several authors have suggested that the preferred scope ordering must be a function of: the quantifiers and their structural position (Reinhart, 1983; May, 1985), the semantics of the main verb (Scha, 1981; van der Does, 1994), the cardinality conditions underlying the main relation (Sher, 1990), or the functional dependency implied by the main verb (Park, 1995; Hobbs, 1996).

2.3.4 Preference Rules in the Resolution of Quantifier Scope

Quantifier scope resolution algorithms developed by computational linguists use syntactically motivated preference rules. These rules do not always work. In this sub section we state some of the rules and show that they do not always work.

Rule:

A quantifier (except for the definite quantifiers) in a relative clause must depend on (i.e., must have a narrow scope relative to) the quantifier of its head noun. (Allen, 1987; Grosz et. al, 1987; Moran, 1988; Moran And Pereira 1992).

For Example:

(14) Several programmers that work for a text retrieval company in Ottawa visited MIT

According to the rule the quantifier 'a' in the noun phrase 'a text retrieval company' (which is in a relative clause) must take a narrow scope relative to the quantifier 'several' in the head noun phrase. A reader of this sentence prefers the reading "there is a text Retrieval Company in Ottawa and several programmers that work for this company visited MIT. Thus giving the quantifier in the relative clause wider scope to the quantifier of its head noun phrase.

Other examples:

(15) Several students that enrolled for an AI course offered by Dr. Moore in university of Princeton attended Dr. Millers seminar on KDD.

(16) Several students that took a psychology course of Dr. Jones at university of Toronto were invited to attended psycholinguistic seminar of Dr. Smith at MIT.

(17) Several families whose members work for a software company in Ottawa were invited for Annual celebration by its president.

It is apparent from the above examples that the scope preference implied by the rule does not always capture an individuals preferred reading. In example (15) based on their knowledge about students enrolling for an AI course at a university, most readers would prefer a reading that there is an AI course offered by Dr. Jones which is taken by several students at Princeton University and these students attended Dr. Miller's seminar on KDD. Examples (16) and (17) are similar in preferred reading

Rule:

A quantifier within a prepositional phrase modifier is scoped wider than the quantified phrase it modifies. (Allen, 1987; Grosz et. al, 1987; Moran, 1988; Moran And Pereira, 1992).

For Example:

(18) John interviewed every president of a major software company.

According to the rule the quantifier 'a' in the prepositional phrase that modifies the quantified phrase 'every president' should take a wider scope over it, thus reading, "a major software company has many president's whom john interviewed". From our world knowledge that major software companies usually have only one president, this reading is not plausible. A reader of this sentence will select a reading where the quantifier in the prepositional phrase takes a narrow scope over the quantified phrase it modifies.

Other Examples:

(19) Its members invite every chairman of a committee.

(20) Every mother of a newborn feeds the baby.

(21) The publisher invited every authors of a novel.

(22) Every director of a movie aims for an Oscar award.

(23) Someone reads every article written in a science magazine.

(24) Every scientist of a country aims for a noble prize.

It is clear from the above examples that the scope preference implied by the rule does not always capture the preferred reading of an individual. In example (19) a reader based on the world knowledge about chairmen of committee would prefer a direct reading of the sentence thus giving the quantifier 'a' in the prepositional phrase a narrow scope over the quantified phrase it modifies. Similar are the examples (20), (21) and (22). In examples (23) and (24) a reader would prefer a direct scope ordering of the sentence to their indirect scope ordering.

Rule:

A quantifier cannot be raised across more than one major clause boundary (Allen, 1987; Moran, 1988; Moran And Pereira, 1992).

For Example:

(25) A patient is admitted in a hospital of every city.

The scope preference implied by the rule does not capture an individuals preferred reading. That is a reader of this sentence will select a scope reading, "every city has a hospital in which a patient is admitted". Thus the quantifier "every" in the prepositional phrase "of every city" is given the widest scope (i.e. raised over more than one major clause boundary).

Other Examples:

(26) John visited a restaurant on a main street of every city.

(27) Several programmers that work for a major software company in every city met their president.

(28) A student in a department of every university applied for graduation.

(29) A restaurant at a beach in every costal city has pleasant sea view.

In all of the above examples the quantifier "every" in the prepositional phrase takes the widest scope thus raised across more than one major clause boundary.

Rule:

There is a strong preference for "each" to outscope other determiners (Moran, 1988; Moran And Pereira, 1992)

Example that breaks this preference:

(30) John visited each chines restaurant on every street of Ottawa.

(31) John visited each country in every continent.

The preference rules (that are syntactically-motivated) fail in some cases and purely linguistic considerations do not always capture some pragmatic and inferential aspects of quantifier scope (Saba & Corriveau, 1999). From the above examples it is clear that this is true and pragmatic information should be taken into account to arrive at a most plausible or preferred reading.

In the next section we discuss an inferential approach to resolution of QSA.

2.4 The QC Algorithm

In the previous section we saw that the computational approaches to resolution of quantifier scope ambiguity face a number of problems. In this section I look at an algorithm proposed by (Saba, 1999; Saba & Corriveau, 2001), that uses commonsense reasoning for the resolution process and aims at deciding on the most plausible scope ordering. This resolution process is two-step. First, by the process of logical entailment plausibility of a scope reading of a sentence involving quantifiers is determined by discarding readings that are inconsistent with some pragmatic considerations. In the process of logical entailment the quantificational constraint (QC), which is a piece of commonsense knowledge (see Saba, 2001) reflecting an individual belief as to how an certain relation R between two concepts C_1 and C_2 can possibly be, is typically, or must be manifested in some possible world, is used to impose a set of conditions on R. Each scope ordering of a sentence involving quantifiers is then tested to see their consistency with the conditions imposed by the QC. Those scope readings that are inconsistent with the conditions imposed by the QC are discarded due to pragmatic considerations and thus this process gives plausible scope reading of a quantified sentence. Secondly, determining the plausibility of a scope reading is not sufficient, a numerical process that determines the ***degree*** of plausibility is devised, as there could be situations where two scope readings can be equally plausible or one of them discarded due to pragmatic considerations or both plausible but only one is more acceptable. The QC algorithm ranks each plausible scope ordering of a sentence involving multiple quantifiers thus selecting the most plausible scope ordering. That is the process of logical entailment gives a yes/no decision on the plausibility of a scope reading and the numerical algorithm gives the degree of plausibility of each scope reading.

The main features of the QC algorithm are:

- It models individual preference in selecting the most plausible scope ordering.
- It explains the transformational puzzle.
- It models complex functional dependencies (and thus explains the problem of 'branching quantifiers')
- It solves the combinatorial explosion problem.

Below we discuss these features in brief.

2.4.1 The QC

In resolving quantifier scope ambiguities the algorithm uses a quantificational constraint (QC) that is assumed to reflect some individual's commonsense belief that manifest in a certain context. It is defined as (Saba & Corriveau 1999):

$$(33)\left(QC(R,C_1,C_2)=\langle m_1,m_2\rangle\right)\equiv \text{def}\left[\begin{array}{l}((\forall s_1\subseteq C_1)((|s_1|=m_1)\supset(\exists s_2=C_2)((|s_2|=m_2)\wedge(R(s_1,s_2)))))\\ \wedge((\forall s_2\subseteq C_2)((|s_2|=m_2)\supset(\exists s_1=C_1)((|s_1|=m_1)\wedge(R(s_1,s_2)))))\end{array}\right]$$

It is assumed that for every relation R between two concepts C_1 and C_2 there is a quantificational constraint $QC(R,C_1,C_2)=\langle m_1,m_2\rangle$ that is defined as in (33) and reflects an individual's belief as to how a certain relation

Can possibly be,

is typically,

Must be

Manifested in some possible world (Saba & Corriveau, 1999).

QC(R, C_1, C_2) is a pair of values < m_1, m_2 > where m_1 is the number of C_1 elements that are in relation R to the same C_2 element, and m_2 is the number of C_2 elements that the same C_1 element is related to by R (Saba & Corriveau, 2001).

2.4.1.1 QC's

In this subsection some example QC's are shown that are assumed to reflect an individuals belief as explained above. What is this piece commonsense knowledge that reflects what an individual believes, which is captured by the quantificational constraint? To explain this we discuss two examples. Consider the *Manufacture* relationship defined between an *AutoCompany* and a *Car* and the *drive* relation defined between a *Person* and a *Car.*

The QC's, are[4]:

(34) $QC(\text{Manafacture, Auto Company, Car}) = \langle 1, \text{many} \rangle$

(35) $QC(\text{Drive, Person, Car}) = \langle 1, 1 \rangle$

These QC's captures what an individual believes that typically an auto company can manufacture many cars (and that the same car can be manufactured by only one auto company), and only one person can drive a car (and same car can be driven by one person). In the next sub section we discuss the effect of modal and temporal aspects on the QC's.

2.4.1.2 Modal and Temporal Aspects

For a detailed explanation as to how the modal and temporal aspects effect the choice of quantifier scope and how they are used in inferring the most plausible scope

[4] QCs are individual, and thus the QCs I suggest here are mine. They simply reflect how I see certain relations manifested in the world we live in.

ordering see (Saba & Corriveau, 1999). Here we only discuss the effect of modal and temporal aspects on the choice of a QC. For example, while it is *necessary* for a car to be manufactured by an auto company it is not *necessary* that a car be driven by a person. Thus, what is implicit in the QCs we suggest is a certain modality. Moreover, when temporal and modal aspects are considered the initially assumed (default) QC's might change based on surface structure information (such as tense). For example, at a specific point in time a person can drive one car and the same car can be driven by only one person. Over time, however, the same car can be driven by many persons, and the same person can drive many cars. Thus in a context implying events occurring over time, the QC(Drive,Person,Car) = <1,1> moves into QC(Drive,Person,many,many).

The table below illustrates the possible QC's for (34) and (35) for the Manufacture and Drive relations between Auto company/Car and Person/Car respectively along temporal and modal dimensions.

#Car (Manufactured by) Auto Company	Is typically Manufactured	Can Possibly be Manufactured	Must be Manufactured
At some specific point in time	1	1	1
At various points in time	1	1	1
# Car (Driven by) Person	Typically Driven By	Can Possibly be Driven by	Must be Driven by
At some specific point in time	?	1	0
At various points in time	?	Many	0

A more detailed illustration of the QCs for the On relation between houses/streets and books/shelves along the temporal and modal dimensions is given in Saba & Corriveau, (1999).

2.4.2 Measuring the Plausibility of a Scope Reading and Determining the Most Plausible Scope Reading

The process of logical entailment gives the plausibility of a scope ordering by using the conditions imposed by the corresponding quantificational constraint. That is a quantificational constraint $QC(R, C_1, C_2) = \langle m_1, m_2 \rangle$ defines two partitions on the sets C_1 and C_2 where subsets in these partitions can be related to each other by R according to the following (Saba & Corriveau, 2001).

$$P_1 = \{(X \subseteq C_1) | (1 \leq X \leq m_1) \wedge (\exists Y \subseteq C_2)((1 \leq Y \leq m_2) \wedge R(X, Y))\} \quad (36)$$

$$P_2 = \{(Y \subseteq C_2) | (1 \leq Y \leq m_2) \wedge (\exists X \subseteq C_1)((1 \leq X \leq m_1) \wedge R(X, Y))\} \quad (37)$$

Here the QC defines some conditions on generating the sets of C_1 and C_2 in the context of the relation R. The logical form of a scope reading is then tested for its consistency with the above conditions imposed by a corresponding quantificational constraint. Consistency gives the plausibility of a scope ordering by discarding scope orderings of a sentence involving quantifiers that are inconsistent with the imposed conditions due to pragmatic considerations. The scope readings are not rejected on logical grounds but instead are rejected on the grounds of inconsistency with some pragmatic considerations (Saba & Corriveau, 1999). The process of logical entailment gives a yes/no decision on the plausibility of a scope reading. Also a process that determines the degree of plausibility of a scope reading is needed to decide which scope reading is most plausible in case of more than one plausible score reading of the same sentence (thus this process obtains the most plausible scope reading). For this the process of logical entailment is extended to a numerical algorithm in (Saba & Corriveau, 1999). The numerical algorithm is discussed in brief below.

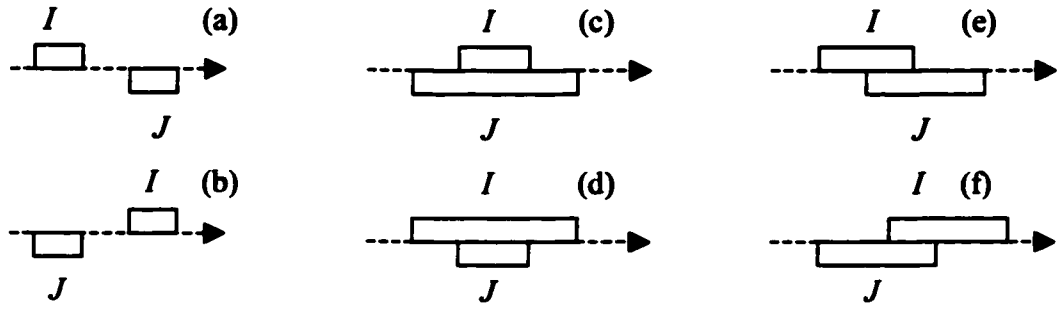
Given a quantificational constraint $QC(R, C_1, C_2) = \langle m_1, m_2 \rangle$ and a sentence

$$(S(NP(DET\ Q_1)(TP[p_1,...,p_m]X))(VP(v\ R)(NP(DET\ Q_2)(TP[q_1,...,q_n]Y))))$$

The degree of plausibility of the direct and indirect readings of a sentence is the number of ways the following inequality holds, respectively (Saba & Corriveau, 2001):

$$(38)\qquad \begin{aligned} r_1 = 1 \le \frac{\text{Range}(Q_2, C_2)}{\text{Range}(Q_1, C_1)} |X| \le \frac{[1,\ldots,m_2]}{[1,\ldots,m_1]} |C_1| \\ r_2 = 1 \le \frac{\text{Range}(Q_1, C_1)}{\text{Range}(Q_2, C_2)} |Y| \le \frac{[1,\ldots,m_1]}{[1,\ldots,m_2]} |C_2| \end{aligned}$$

The inequalities r_1 and r_2 each give rise to intervals I and J. The degree of plausibility of a scope reading can be determined by comparing the intervals I and J to check the number of ways an interval I (strictly) precedes an interval J. In computing this value the interesting situations are shown below. Here we briefly summarize the interesting situations of I and J and the process of computing these situations that is explained in detail in (Saba & Corriveau, 2001).



In (a) the plausibility measure is 1.0, and in (b) it is 0. In (c) through (f) the plausibility is somewhere between 0 and 1.0. Every value in the range of I and J are compared to determine the measure of plausibility that could be any value in the range of 0 to 1. The function p (I, J) computes the measure of plausibility using the formula:

$$p(I,J) = \Pr(I,J)|J| + \sum_{i=1}^{|I \cap J|} \left(|J| - \Pr(J,I) - i + 1\right) |I \cap J| \left(\frac{1}{|I|\ |J|}\right)$$

Where,

$$\Pr(I,J) = |\{i \in I \mid i < J_{min}\}|$$

Here Pr (I, J) is the number of elements in the interval I that are strictly less than the minimum value in J.

2.4.3 Transformational Puzzle: An Illustration with an Example

In this section we show with an example of how the inequalities in (38) are used to determine the degree of plausibility of scope ordering and explain transformational puzzle.

Consider the following example:

Every auto company manufactured a car (39)

We use the QC in (34) and the inequalities in (38) to obtain the degree of plausibility of scope orderings as follows:

$$r_1 \equiv 1 \leq \frac{\text{Range}(Q_2, C_2)}{\text{Range}(Q_1, C_1)} |X| \leq \frac{[1,\ldots,m_2]}{[1,\ldots,m_1]} |C_1|$$

$$\equiv 1 \leq \frac{1}{all} |\text{auto company}| \leq \frac{m_2=[1,\ldots,all]}{m_1=[1]} |\text{auto company}|$$

$$\equiv 1 \leq \frac{1}{all} \leq \frac{m_2=[1,\ldots,all]}{m_1=[1]}$$

$$\equiv \text{True}, \forall m_1, m_2$$

$$r_2 \equiv 1 \leq \frac{\text{Range}(Q_1, C_1)}{\text{Range}(Q_2, C_2)} |Y| \leq \frac{[1,\ldots,m_1]}{[1,\ldots,m_2]} |C_2|$$

$$\equiv 1 \leq \frac{all}{1} |\text{Car}| \leq \frac{m_1=[1]}{m_2=[1,\ldots,all]} |\text{Car}|$$

$$\equiv 1 \leq \frac{all}{1} \leq \frac{m_1=[1]}{m_2=[1,\ldots,all]}$$

$$\equiv \text{False}, \forall m_1, m_2$$

This numerical measure gives the degree of plausibility of each scope ordering and shows that it is the direct reading that is most plausible. The main point is that, no matter how (39) is uttered (i.e., in active or passive form), the only plausible reading is the one implying that every car is manufactured by some (different) auto company. This

is a situation where the transformation between the active and passive forms is said to be meaning preserving, since regardless of which form is uttered one reading (and thus one meaning) is selected (Saba & Corriveau, 2001).

2.4.4 The Linguistic Context

In (38) we see that r_1 and r_2 are functions of the sets C_1 and C_2 and also X and Y. Noun Phrases do not always refer to simple nouns but can involve any number of adjectives, prepositional phrases, relative clauses, etc. thus depending on the linguistic context, the size of X and Y could be a small fraction of the original concept (Saba & Corriveau, 1999). Though linguistic context plays a crucial role in the apparent scope preference we need to note that the inequalities in (38) are not only a function of the linguistic context but also a function of conditions imposed by the quantificational constraint.

For example consider:

Every newspaper advertised a commodity. (40)

A relevant QC is QC (Advertised, newspaper, commodity) = $<1^+, 1^+>$.

$$r_1 \equiv 1 \le \frac{1}{all}|\text{Newspaper}| \le \frac{m_2 = [1,...,all]}{m_1 = [1,...,all]}|\text{Newspaper}|$$

$$\equiv 1 \le \frac{1}{all} \le \frac{m_2 = [1,...,all]}{m_1 = [1,...,all]}$$

$$\equiv \text{True}, \forall m_1, m_2$$

$$r_2 \equiv 1 \le \frac{all}{1}|\text{Commodity}| \le \frac{m_1 = [1,...,all]}{m_2 = [1,...,all]}|\text{Commodity}|$$

$$\equiv 1 \le \frac{all}{1} \le \frac{m_1 = [1,...,all]}{m_2 = [1,...,all]}$$

$$\equiv \text{True when } (m_1, m_2) = (all, 1) \text{ only}$$

From above we can cancel out |X| and $|C_1|$ and similarly for |Y| and $|C_2|$, since the quantified concepts in the sentence where not modified. When the quantified concepts are

modified (adjectives, relative clauses, prepositional phrases, etc.) in a sentence there could be a change in the available scope preferences, and this is precisely what is captured by the inequalities given in (5) (Saba & Corriveau, 2001). Consider the following example:

Every American newspaper advertised a commodity of Johnson& Johnson Pvt Ltd. (41)

$$r_1 \equiv 1 \leq \frac{1}{all} |\text{American Newspaper}| \leq \frac{m_2 = [1,\ldots,all]}{m_1 = [1,\ldots,all]} |\text{Newspaper}|$$

$$\equiv 1 \leq \frac{1}{all} \leq \frac{m_2 = [1,\ldots,all]}{m_1 = [1,\ldots,all]}$$

$$\equiv \text{True}, \forall m_1, m_2$$

$$r_2 \equiv 1 \leq \frac{all}{1} |\text{Commodity of Johnson \& Johnson Pvt Ltd}| \leq \frac{m_1 = [1,\ldots,all]}{m_2 = [1,\ldots,all]} |\text{Commodity}|$$

$$\equiv 1 \leq \frac{all}{1} \leq \frac{m_1 = [1,\ldots,all]}{m_2 = [1,\ldots,all]}$$

$$\equiv \text{True for m combinations of } [1,\ldots,all]/[1,\ldots,all]$$

From examples (40) and (41) we see that the direct readings are very plausible where as the indirect reading of (41) becomes quiet plausible due to the added linguistic context. That is, the overall context in (41) also makes the possibility of a single commodity being implied by the sentence quiet plausible, and particularly if the sentence was uttered in the passive form. Overall the entire inferencing process is defeasible that is initial inferences might be retracted.

2.4.5 Functional Dependencies and Branching Quantifiers

The relationship between functional dependencies and quantifier scope and the explanation of branching quantifiers that are accounted for in this model are illustrated below with simple examples.

$$\left\{\begin{array}{c}\text{(42) A car}\\ \text{(43) A car of every model}\\ \text{(44) A car of every model}\end{array}\right\} \text{is advertised in} \left\{\begin{array}{c}\text{a magazine}\\ \text{a magazine}\\ \text{a magazine of every country}\end{array}\right\}$$

A reasonable quantificational constraint for (42) is

$$QC(\text{Advertised, Car, Magazine}) = \langle 1^{+}, 1^{+} \rangle$$

Using this QC and applying the inequality rules in (38) on (42) shows that both readings of (42) are quiet plausible, and thus the direct reading of (9) should be preferred (Saba & Corriveau, 2001) as we stick with the intention of the author.

In (43) we have a relation '**advertised**' between two quantified NP's, one of which (the head NP) involves scope ambiguity. Thus (43) is processed as

Scope (A car of every model is advertised in a magazine)

= Scope_1 (Scope_2 (A car of every model) is advertised in a magazine)

$$\text{Scope}_1 \left[\begin{array}{c}\text{Every (...model m, a car of m...) is advertised in}\\ \text{a (...magazine...)}\end{array}\right]$$

Here the relevant QC for the head noun phrase is $QC(\text{Of, Car, Model}) = \langle 1^{+}, 1 \rangle$ applying Scope_2 (**Scope** is a second-order function that repeatedly applies the scope rules of (38) on constituents involving a binary relation between two quantified noun phrases) on 'a car of every model' we get the indirect reading as most plausible. Thus 'a car' might potentially refer to many (as opposed to a single) cars, one of every model. Now applying the numerical scope function of (38) to Scope_1 we get the direct reading as acceptable. The final reading of (44) is 'every car of some model is advertised in some magazine'.

We note that unlike (43), the addition of PP 'of every model' to the head noun phrase 'a car' makes the indirect reading of (44) implausible. That is, while it is quiet plausible for a magazine to advertise a specific car, it is not likely that a magazine can advertise a (specific) car, which is of every model. Unlike 'a car' therefore 'a car of every model' does not refer to a single car, but to several (one of every model). Since the scope

rules of (38) are sensitive to the relative size of the quantified sets, these subtle functional dependencies can be easily accounted for in this method (Saba & Corriveau, 2001).

The example (44) explains the conditions under which the quantifiers branch. In this example the two pairs of quantifiers are independent with regard to scope.

The intermediate relation that glues together the entire sentence suggests that once the scope orderings of both the pairs has been decided, the two pairs are independent (Saba & Corriveau, 2001). The scooping of the sentence (44) proceeds as follows:

Scope_0(A car of every model is advertised in a magazine of every country)

$$\text{Scope}_0\begin{pmatrix}\text{Scope}_1\,(\text{A car of every model})\,\text{is advertised in}\\ \text{Scope}_2\,(\text{a magazine of every country})\end{pmatrix}$$

$$\text{Scope}_0\begin{pmatrix}\text{Every}\,(\ldots\,\text{car}\,\ldots)\,\text{is advertised in}\\ \text{Every}\,(\ldots\text{magazine}\ldots)\end{pmatrix}$$

The QC's used are the following:

$$QC(\text{Of}, \text{Car}, \text{Model}) = \langle 1^+, 1 \rangle \quad (45)$$

$$QC(\text{Of}, \text{Magazine}, \text{Country}) = \langle 1^+, 1 \rangle \quad (46)$$

$$QC(\text{Advertised}, \text{Car}, \text{Magazine}) = \langle 1^+, 1^+ \rangle \quad (47)$$

Applying the numerical scope algorithm (38) in Scope_1 and Scope_2 using the QC's (45) and (46) respectively results in reversing the scope ordering (i.e., the indirect reading is the selected scope ordering) in both cases.

Once the scopes of the constituent have been determined, the scope of the entire sentence can be determined in two equivalent ways; this is a situation where the two readings are equally plausible. The reason the two pairs of quantifiers are branching in this case, is a function of the intermediate relation and the quantifiers that result in noun phrase of each constituent (which in turn is a function of the scooping at the constituent level) (Saba & Corriveau, 2001). Now if we replace 'every model' by 'a model' the pair of quantifiers would not be branching and one scope ordering would be accepted at the outermost level.

2.5 The Problem of Negation

A sentence can have negation words like 'No', 'Not all', etc., in the determinar position of a noun phrase. This causes scope ambiguity. The QC algorithm however can not directly handle negation. The problem here is how do we treat the negation words? For instance if we treat the negation word 'No' with a value zero then the QC algorithm has cases of $\left[\frac{0}{0}\right]$: undefined, $\left[\frac{\text{positive integer}}{0}\right]$: infinity, for which we need to apply a method like limits and other choice is to treat 'No' as $\lambda P \lambda Q[(\forall x)(P(x) \rightarrow \neg Q(x))]$ which is the meaning of 'No' as in (Montague, 1973). More details about Negation and the approach taken are discussed in the next chapter.

Chapter 3

Extending the QC Algorithm to Account for Negation

3.1 Introduction

Different scopes that can be assigned to various quantified noun phrases in a sentence are one major cause of ambiguity. A noun phrase can involve words like 'No', 'Not all', etc. Some cases where negation occurs in a sentence are (a) words like 'No', 'not all' in the determiner position of a noun phrase (b) words like 'not' negating the verb phrase. That is, for a given sentence (4.1) 'No', 'Not all' can appear as determiners Q_1 or Q_2 or both also 'not' can negate the verb phrase.

(3.1) $S=[_S[NP[_{Det}\,Q_1]X]VP[_V R]NP[_{Det}\,Q_2]Y]$

Various factors are involved in determining the preferred scope ordering of quantifiers. Below are some sentences with either 'No' occurring in the noun phrase or 'Not' negating the verb phrase. From these examples we see that 'No' can take a narrow scope or a wide scope. In (4.2) 'No' takes a wide scope where as in (4.3) 'No' takes a narrow scope.

'No' appearing as Q_1.

(3.2) No student won an award.

'No' appearing as Q_2.

(3.3) A student won no award.

'Not' negating verb phrase.

(3.4) a student did not win any award.

The possible scope readings for (3.2) are (3.2.I) none of the students won any award (3.2.II) an award is won by none of the students and the possible scope readings of

(3.3) are (3.3.I) a student won none of the awards (3.3.II) none of the awards are won by any student and the possible scope readings of (3.4) are (3.4.I) a specific student did not win any award and (3.4.II) take any award some student did not win.

A reader of a sentence (3.1) has a Quantificational constraint in mind that is a cause for preferring a scope reading (Saba & Corriveau, 1999). The preferred scope reading in case of (3.2) is (3.2.I) and in case of (3.3) is (3.3.I) and in case of (3.4) is (3.4.I). The numerical algorithm in (2.38) cannot directly handle negation. We investigate two approaches as to how negation can be handled by the numerical algorithm in (2.38).

3.2 Approach

The linguistic 'No' is eliminated as a quantifier and is instead pushed to the relation as a unary operator $\neg$ (meaning: 'not'). First, the meaning of 'No' as given as

$$\text{No} \equiv \lambda P \lambda Q [(\forall x)(P(x) \rightarrow \neg Q(x))]$$

For example consider the following:

(4.5) No student walks $\Rightarrow (\forall s)(\text{student}(s) \rightarrow \neg \text{Walks}(s))$

(4.6) No student $\Rightarrow \lambda Q [(\forall s)(\text{Student}(s) \rightarrow \neg Q(s))]$

(4.7) No $\Rightarrow \lambda P \lambda Q [(\forall s)(P(s) \rightarrow \neg Q(s))]$

(4.5) Is the straightforward translation of 'No student walks' into FOPL. Using lambda abstraction, abstracting from the formula the property of "walking", the result is a function (4.6) that takes a property and returns true if the property is true of every student and false otherwise. By another lambda abstraction over the property of "being a student"

results in a function (4.7) that takes two properties and returns a true value, which is the meaning of 'No'.

The straightforward translation of (4.2) into FOPL is

$$(4.7)\quad \begin{array}{l}(\forall s)(\text{student}(s) \rightarrow \neg(\exists a)(\text{award}(a) \wedge \text{Win}(s, a))) \\ (\forall s)(\text{student}(s) \rightarrow (\forall a)(\text{award}(a) \wedge \neg\, \text{Win}(s, a)))\end{array}$$

Similarly for (4.3)

$$(4.8)\quad (\exists s)(\text{student}(s) \wedge (\forall a)(\text{award}(a) \rightarrow \neg\, \text{Win}(s, a)))$$

Using techniques such as quantifier rising or quantifying in we obtain the other scope reading of (4.2)-(4.4). That is, for sentence of the form (4.1) we have the following scope readings.

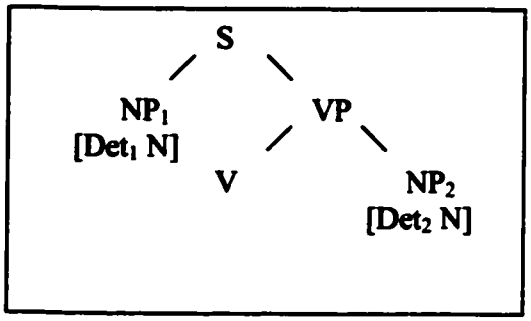


Figure: 1. The syntactic structure of a sentence.

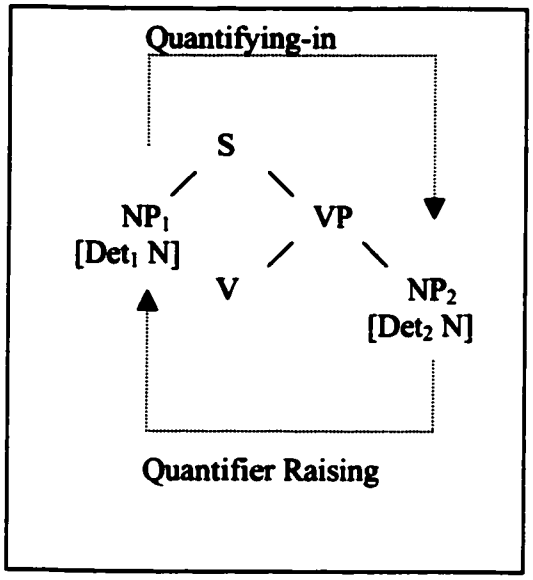


Figure: 2. The mechanisms of quantifier rising and quantifying in.

Notation
Det_1 = a, every, all, any, no, etc.,
Det_2 = a, every, all, any, no, etc.,
V = V or ¬ V

Table: 1. Description of notations used in figure: 2.

The other scope readings of (4.2) and (4.3) are (4.9) and (4.10) respectively:

$(4.9)(\exists a)(\text{award}(a) \wedge (\forall s)(\text{student}(s) \rightarrow \neg \text{Win}(a, s)))$

$(4.10)(\forall a)(\text{award}(\ ) \rightarrow (\forall s)(\text{student}(s) \rightarrow \neg \text{Win}(a, s)))$

The following table shows the scope reading of sentence (3.2) – (3.4).

(3.2)	Direct Reading	$(\forall s)(\text{student}(s) \rightarrow (\forall a)(\text{award}(a) \wedge \neg \text{Win}(s, a)))$
	Indirect Reading	$(\exists a)(\text{award}(a) \wedge (\forall s)(\text{student}(s) \rightarrow \neg \text{Win}(a, s)))$
(3.3)	Direct Reading	$(\exists s)(\text{student}(s) \wedge (\forall a)(\text{award}(a) \rightarrow \neg \text{Win}(s, a)))$
	Indirect Reading	$(\forall a)(\text{award}(\ ) \rightarrow (\forall s)(\text{student}(s) \rightarrow \neg \text{Win}(a, s)))$
(3.4)	Direct Reading	$(\exists s)(\text{student}(s) \rightarrow (\forall a)(\text{award}(a) \wedge \neg \text{Win}(s, a)))$
	Indirect Reading	$(\forall a)(\text{award}(\ ) \rightarrow (\exists s)(\text{student}(s) \rightarrow \neg \text{Win}(a, s)))$

Table: 2. Logical Form of Scope readings for the example sentences.

Now substituting values into the numerical algorithm in (2.38) we get,

For the example of (4.2)

$$r_1 = 1 \leq \frac{[Q_2 = 1,\ldots,all]}{[Q_1 = all]}|Student| \leq \frac{[m_2 = 1,\ldots,all]}{[m_1 = 1]}|Student|$$

$$r_2 = 1 \leq \frac{[Q_1 = 1,\ldots,all]}{[Q_2 = 1]}|Award| \leq \frac{[m_1 = 1]}{[m_2 = 1,\ldots,all]}|Award|$$

The numerical process selects r_1 as the most likely reading and this is the preferred reading of (4.2).

For the example of (4.3)

$$r_1 = 1 \leq \frac{[Q_2 = 1,\ldots,all]}{[Q_1 = 1]}|Student| \leq \frac{[m_2 = 1,\ldots,all]}{[m_1 = 1]}|Student|$$

$$r_2 = 1 \leq \frac{[Q_1 = 1,\ldots,all]}{[Q_2 = all]}|Award| \leq \frac{[m_1 = 1]}{[m_2 = 1,\ldots,all]}|Award|$$

The numerical process selects r_1 as the most likely reading and this is the preferred reading of (4.3).

For the example of (4.4)

$$r_1 = 1 \leq \frac{[Q_2 = 1,...,all]}{[Q_1 = 1]}|student| \leq \frac{[m_2\ 1,...,all]}{[m_1 = 1]}|student|$$

$$r_2 = 1 \leq \frac{[Q_1 = 1]}{[Q_2 = 1,...,all]}|award| \leq \frac{[m_1 = 1]}{[m_2 = 1,...,all]}|award|$$

The numerical process selects r_1 and r_2 as the most likely reading, and by the heuristic of selecting the direct reading in such cases, r_1 is chosen as the preferred reading.

In both the approaches the interest is in checking the scope readings preferred by the numerical algorithm for sentence (3.1) with 'No' on Q_1 or Q_2 and 'Not' negating the verb phrase. The results obtained are either accepted (i.e., select the scope reading that is likely) or reversed (i.e., select the scope reading that is less likely).

3.3 Syntactic Constituents and Types of Quantificational Constraints

Various syntactic constituents and types of quantificational constraints are considered in this approach. The following subsection gives the list.

3.3.1 Choice of the QC's

The definition of $QC(R, C_1, C_2) = \langle m_1, m_2 \rangle$ is given in chapter 2. $QC(R, C_1, C_2)$ Is a pair of values $\langle m_1, m_2 \rangle$ where m_1 and m_2 are assumed to be either constant numeric values (e.g., one, two, at least two, at most three), standard linguistic quantifiers (e.g., a, some, every), or context-dependent (fuzzy) linguistic quantifiers (e.g., few, several, many) (Saba, 1999).

Sentences of the form (4.1) whose QC's have the following values for m_1 and m_2, are chosen.

$QC(R, C_1, C_2) = \langle 1, many \rangle$
$QC(R, C_1, C_2) = \langle 1, few \rangle$
$QC(R, C_1, C_2) = \langle few, many \rangle$
$QC(R, C_1, C_2) = \langle few, few \rangle$
$QC(R, C_1, C_2) = \langle many, many \rangle$
$QC(R, C_1, C_2) = \langle many, few \rangle$
$QC(R, C_1, C_2) = \langle few, 1 \rangle$
$QC(R, C_1, C_2) = \langle many, 1 \rangle$

3.3.2 Choice of Quantifiers

For a sentence of the form (4.1) negation can appear in the determiner positions Q_1 or Q_2 or both or on the relation R. Here sentences with negation appearing on either Q_1 or Q_2 or on the relation are chosen. For a sentence of the form (4.1) the format of representation chosen is $Q_1\ C_1\ R\ Q_2\ C_2$.

Where Q_1 and Q_2 are quantifiers, C_1 and C_2 are concepts and R is a relation. The concepts C_1 and C_2 can have modifiers like adjectives, prepositional phrases, relative clause, etc., (i.e., $Q_1\ [p_1, p_2, \ldots, p_n]C_1\ R\ Q_2\ [q_1, q_2, \ldots, q_m]C_2$). The following are the sentences chosen for our purpose that have quantifiers varying on Q_1 and Q_2 shown below.

$$\text{No } C_1\ R\ Q_2\ C_2 \equiv \begin{bmatrix} \text{No } C_1\ R\ \text{a } C_2 \\ \text{No } C_1\ R\ \text{every } C_2 \\ \text{No } C_1\ R\ \text{all } C_2 \\ \text{No } C_1\ R\ \text{any } C_2 \end{bmatrix}$$

Where,

Q_1 = No
Q_2 = a, every, all or any

Example sentences:

No Engineer is employed in $\left\{\begin{matrix}\text{a}\\\text{every}\\\text{all}\\\text{any}\end{matrix}\right\}$ company(s).

with linguistic context

No software Engineer is employed in $\left\{\begin{matrix}\text{a}\\\text{every}\\\text{all}\\\text{any}\end{matrix}\right\}$ company(s) in ottawa.

$$Q_1\ C_1\ R\ \text{No}\,C_2 \equiv \begin{bmatrix}\text{A}\ C_1\ R\ \text{no}\,C_2\\ \text{Every}\ C_1\ R\ \text{no}\,C_2\\ \text{All}\ C_1\ R\ \text{no}\,C_2\end{bmatrix}$$

Where,

Q_1 = a, every or all.
Q_2 = No

Example sentences:

$\left\{\begin{matrix}\text{A}\\\text{Every}\\\text{All}\end{matrix}\right\}$ student submitted no paper.

with linguistic context

$\left\{\begin{matrix}\text{A}\\\text{Every}\\\text{All}\end{matrix}\right\}$ MIT student submitted no paper to ACL'2002.

$$A\ C_1\ \neg R\ Q_2\ C_2 \equiv \begin{bmatrix} A\ C_1\ \neg R\ \text{all}\ C_2 \\ A\ C_1\ \neg R\ \text{any}\ C_2 \end{bmatrix}$$

Where,

$Q_1 = A$
Q_2 = all or any.

Example Sentences:

A student did not attend $\left\{\begin{matrix}\text{all} \\ \text{any}\end{matrix}\right\}$ seminar.

or with linguistic context

A graduate student did not attend $\left\{\begin{matrix}\text{all} \\ \text{any}\end{matrix}\right\}$ AI seminar.

Every $C_1\ \neg R\ \text{any}\ C_2$

GOAL: The goal is to extend quantifier scope resolution algorithm to handle negation by making observations over the selected scope preference made by the QC algorithm for the above-mentioned choices of quantificational constraints and combination of positioning negation and quantifiers for a sentence of the form (4.1).

In the next section the results as per the goal are shown.

3.4 Results

The following are the results obtained. Here above-mentioned choices for a sentence are considered (i.e., choice of negation, quantifiers, QC's, etc.,).

3.4.1 Type of Sentence $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ \text{R A}\,[q_1, q_2, \ldots, q_n]C_2$

The results for a sentence of the type $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ \text{R A}\,[q_1, q_2, \ldots, q_n]C_2$ are shown below. Sentences of this type with QC's mentioned in section 3.3.1 are considered:

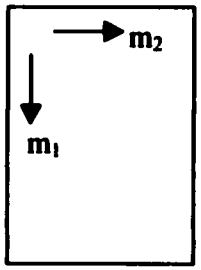
NO/A	(1,m)			WITH
	S		No graduate student of U of W won a NSERC award.	
	D	✓	$(\forall s)(\text{graduate student of U Of W}(s) \rightarrow (\forall a)(\text{NSERC award}(a) \rightarrow \neg \text{win}(s, a)))$	
	I		$(\exists a)(\text{NSERC award}(a) \wedge (\forall s)(\text{graduate student of U of W}(s) \rightarrow \neg \text{win}(s, a)))$	
	r_1	↑	$r_1 = \frac{[1, \ldots, \text{all}_{\text{NSERC Award}}]}{[\text{all}_{\text{graduate student of U of W}}]} \lvert \text{graduate student of U of W} \rvert \leq \frac{[1, \ldots, \text{all}_{\text{award}}]}{[1]} \lvert \textit{student} \rvert$	
	r_2	↙	$r_2 = \frac{[1, \ldots, \text{all}_{\text{graduate student of U of W}}]}{[1]} \lvert \text{NSERC award} \rvert \leq \frac{[1]}{[1, \ldots, \text{all}_{\text{award}}]} \lvert \textit{award} \rvert$	
	(1, f)			WITH
	S		No professor at U of W advises a PhD student.	
	D	✓	$(\forall p)(\text{professor at U Of W}(p) \rightarrow (\forall s)(\text{PhD student}(s) \rightarrow \neg \text{advise}(p, s)))$	
	I		$(\exists s)(\text{PhD student}(s) \wedge (\forall p)(\text{professor at U Of W}(p) \rightarrow \neg \text{advised by}(s, p)))$	
	r_1	↑	$r_1 = \frac{[1, \ldots, \text{all}_{\text{PhD student}}]}{[\text{all}_{\text{professor at U Of W}}]} \lvert \text{professor at U Of W} \rvert \leq \frac{[1, \ldots, \text{all}_{\text{student}}]}{[1]} \lvert \text{professor} \rvert$	
	r_2	↙	$r_2 = \frac{[1, \ldots, \text{all}_{\text{professor at U Of W}}]}{[1]} \lvert \text{PhD student} \rvert \leq \frac{[1]}{[1, \ldots, \text{all}_{\text{student}}]} \lvert \text{PhD student} \rvert$	

	(f,m)		No MIT student submitted a paper on quantification.	WITH
	S		$(\forall s)(\text{MIT student}(s) \rightarrow (\forall p)(\text{paper on quantification}(p) \rightarrow \neg \textit{submit}(s, p)))$	
	D	✓	$(\exists p)(\text{paper on quantification}(p) \wedge (\forall s)(\text{MIT student}(s) \rightarrow \neg \text{submitted by}(p, s)))$	
	I		$r_1 = \frac{\lfloor 1, \ldots, \text{all}_{\text{paper on quantification}} \rfloor}{\lceil \text{all}_{\text{MIT student}} \rceil} \lvert \text{MIT student} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{paper}} \rfloor}{\lceil 1, \ldots, \text{few}_{\text{student}} \rceil} \lvert \textit{student} \rvert$	
	r_1	↑		
	r_2	↙	$r_2 = \frac{\lfloor 1, \ldots, \text{all}_{\text{MIT student}} \rfloor}{\lceil 1 \rceil} \lvert \text{paper on quantifica tion} \rvert \leq \frac{\lfloor 1, \ldots, \text{few}_{\text{student}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{paper}} \rceil} \lvert \textit{paper} \rvert$	
	(m, m)		No science student at university of Windsor attended an AI seminar of John McCarthy on robots.	WITH
	S		$(\forall s)\left(\begin{array}{l}\text{science student at university of Windsor}(s) \rightarrow \\ (\forall r)\left(\begin{array}{l}\text{AI seminar of John McCarthy on robots}(r) \\ \rightarrow \neg \textit{attended}(s, r)\end{array}\right)\end{array}\right)$	
	D	✓		
	I	✓	$(\exists r)\left(\begin{array}{l}\text{AI seminar of John McCarthy on robots}(r) \wedge \\ (\forall s)(\text{science student at university of Windsor}(s) \rightarrow \neg \text{visited by}(r, s))\end{array}\right)$	
	r_1	↑	$r_1 = \frac{\lfloor 1, \ldots, \text{all}_{\text{AI seminar of John McCarthy on robots}} \rfloor}{\lceil \text{all}_{\text{science student at university of Windsor}} \rceil} \lvert \text{science student at university of Windsor} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{seminar}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{student}} \rceil} \lvert \text{student} \rvert$	
	r_2	↙	$r_2 = \frac{\lfloor 1, \ldots, \text{all}_{\text{science student at university of Windsor}} \rfloor}{\lceil 1 \rceil} \lvert \text{AI seminar of John McCarthy on robots} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{student}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{seminar}} \rceil} \lvert \text{seminar} \rvert$	
	(m, f)		No one in Canada read a European newspaper.	WITH
	S		$(\forall c)(\text{Canadian person}(c) \rightarrow (\forall n)(\text{european news paper}(n) \rightarrow \neg \textit{read}(c, n)))$	
	D	✓	$(\exists n)(\text{european newspaper}(n) \wedge (\forall c)(\text{Canadian person}(c) \rightarrow \neg \text{read by}(n, c)))$	
	I	✓		

	r_1	↑	$r_1 = \frac{\lfloor 1,\ldots, \text{all}_{\text{european news paper}} \rfloor}{\lceil \text{all}_{\text{canadian person}} \rceil} \lvert \text{canadian person} \rvert \leq \frac{\lfloor 1,\ldots, \text{few}_{\text{news paper}} \rfloor}{\lceil 1,\ldots, \text{all}_{\text{person}} \rceil} \lvert \textit{person} \rvert$	
	r_2	↙	$r_2 = \frac{\lfloor 1,\ldots, \text{all}_{\text{canadian person}} \rfloor}{1} \lvert \text{european newspaper} \rvert \leq \frac{\lfloor 1,\ldots, \text{all}_{\text{person}} \rfloor}{\lceil 1,\ldots, \text{few}_{\text{news paper}} \rceil} \lvert \textit{newspaper} \rvert$	
	(f,1)			WITH
	S		No faculty member at university of Windsor is chairing an AI conference in Ottawa.	
	D	✓	$(\forall f)(\text{faculty at university of Windsor } (f) \rightarrow (\forall c)(\text{AI conference in Ottawa } (c) \rightarrow \neg \textit{chairing } (f, c)))$	
	I	✓	$(\exists c)(\text{AI conference in Ottawa } (c) \wedge (\forall f)(\text{faculty member at university of Windsor } (f) \rightarrow \neg \text{chaired by } (c, f)))$	
	r_1	↑	$r_1 = \frac{\lfloor 1,\ldots, \text{all}_{\text{AI conference in Ottawa}} \rfloor}{\lceil \text{all}_{\text{faculty member at university of Windsor}} \rceil} \lvert \text{faculty member at university of Windsor} \rvert \leq \frac{\lfloor 1 \rfloor}{\lceil 1,\ldots, \text{few}_{\text{faculty}} \rceil} \lvert \text{faculty} \rvert$	
	r_2	↗	$r_2 = \frac{\lfloor 1,\ldots, \text{all}_{\text{faculty member at university of Windsor}} \rfloor}{\lceil 1 \rceil} \lvert \text{conference in Ottawa} \rvert \leq \frac{\lfloor 1,\ldots, \text{few}_{\text{faculty}} \rfloor}{\lceil 1 \rceil} \lvert \text{conference} \rvert$	
	(m, 1)			WITH
	S		No software engineer works for a company in Ottawa.	
	D	✓	$(\forall e)(\text{software engineer } (s) \rightarrow (\forall c)(\text{company in Ottawa } (c) \rightarrow \neg \text{works for } (e, c)))$	
	I	✓	$(\exists c)(\text{company in Ottawa } (c) \wedge (\forall e)(\text{software engineer } (e) \rightarrow \neg \text{works for } (e, c)))$	
	r_1	↙	$r_1 = \frac{\lfloor 1,\ldots, \text{all}_{\text{company in Ottawa}} \rfloor}{\lceil \text{all}_{\text{software engineer}} \rceil} \lvert \text{software engineer} \rvert \leq \frac{\lfloor 1 \rfloor}{\lceil 1,\ldots, \text{all}_{\text{engineer}} \rceil} \lvert \text{engineer} \rvert$	
	r_2	↑	$r_2 = \frac{\lfloor 1,\ldots, \text{all}_{\text{software engineer}} \rfloor}{1} \lvert \text{company in Ottawa} \rvert \leq \frac{\lfloor 1,\ldots, \text{all}_{\text{engineer}} \rfloor}{\lceil 1 \rceil} \lvert \text{company} \rvert$	

	(1,1)		No MIT student is giving an AI seminar on robotics.	WITH
	S		$(\forall s)(\text{student}(s) \rightarrow (\forall r)(\text{seminar on robotics}(r) \rightarrow \neg \text{giving}(s, r)))$	
	D	✓	$(\exists r)(\text{AI seminar}(r) \wedge (\forall s)(\text{MIT student}(s) \rightarrow \neg \text{given by}(r, s)))$	
	I	✓	$r_1 = \frac{[1,\ldots,\text{all}_{\text{AI seminar on robotics}}]}{[\text{all}_{\text{MIT student}}]} \lvert\text{MIT student}\rvert \leq \frac{[1]}{[1]} \lvert\text{student}\rvert$	
	r_1	↑		
	r_2	↙	$r_2 = \frac{[1,\ldots,\text{all}_{\text{MIT student}}]}{1} \lvert\text{AI seminar on robotics}\rvert \leq \frac{[1]}{[1]} \lvert\text{seminar}\rvert$	

Table 3.1a: Results for sentences of the type $\text{No}[p_1, p_2, \ldots, p_n]C_1 \text{ R A } [q_1, q_2, \ldots, q_n]C_2$ with examples.



No/A	[m_2=1]	[Few]	[Many]
[m_1=1]	1	1	1
[Few]	1e		1
[Many]	1	1	1

Table 3.1b: Results for the example sentences of the type $\text{No}[p_1, p_2, \ldots, p_n]C_1 \text{ R A } [q_1, q_2, \ldots, q_n]C_2$ are depicted in the table.

3.4.2 Type of Sentence $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ \text{R Every}\,[q_1, q_2, \ldots, q_n]C_2$

The results for a sentence of the type $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ \text{R Every}\,[q_1, q_2, \ldots, q_n]C_2$ are shown below. Sentences of this type with QC's mentioned in section 3.3.1 are considered:

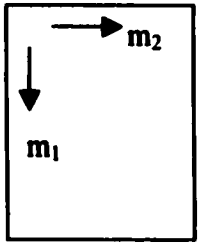
<table>
<tr><td rowspan="8">No/Every</td><td>(1,m)</td><td></td><td rowspan="5">No university admits every prospective applicant.
$(\forall u)(\text{university}(u) \rightarrow (\exists a)(\text{prospective applicant}\ (a) \rightarrow \neg \text{admit}\ (u, a)))$
$(\forall a)(\text{prospective applicant}\ (a) \rightarrow (\forall u)(\text{university}\ (u) \rightarrow \neg \text{admited by}\ (a, u)))$
$r_1 = \frac{[1]}{[\text{all}_{\text{university}}]}|\text{university}| \leq \frac{[1,\ldots,\text{all}_{\text{applicant}}]}{[1]}|\text{university}|$
$r_2 = \frac{[1,\ldots,\text{all}_{\text{university}}]}{[\text{all}_{\text{prospective applicant}}]}|\text{prospective applicant}| \leq \frac{[1]}{[1,\ldots,\text{all}_{\text{applicant}}]}|\textit{applicant}|$</td><td rowspan="5">WITH</td></tr>
<tr><td>S</td><td></td></tr>
<tr><td>D</td><td>✓</td></tr>
<tr><td>I</td><td></td></tr>
<tr><td>r_1</td><td>↑</td></tr>
<tr><td>r_2</td><td>↙</td><td></td></tr>
<tr><td>(1, f)</td><td></td><td rowspan="5">No professor advises every student.
$(\forall p)(\text{professor}\ (p) \rightarrow (\exists s)(\text{student}\ (s) \rightarrow \neg \text{advise}\ (p, s)))$
$(\forall s)(\text{student}\ (s) \rightarrow (\forall p)(\text{professor}\ (p) \rightarrow \neg \text{advised by}\ (s, p)))$
$r_1 = \frac{[1]}{[\text{all}_{\text{professor}}]}|\text{professor}| \leq \frac{[1,\ldots,\text{few}_{\text{student}}]}{[1]}|\text{professor}|$
$r_2 = \frac{[1,\ldots,\text{all}_{\text{professor}}]}{[\text{all}_{\text{student}}]}|\text{student}| \leq \frac{[1]}{[1,\ldots\text{few}_{\text{student}}]}|\text{student}|$</td><td rowspan="5">WITH</td></tr>
<tr><td>S</td><td></td></tr>
<tr><td>D</td><td>✓</td></tr>
<tr><td>I</td><td></td></tr>
<tr><td>r_1</td><td>↑</td></tr>
<tr><td>r_2</td><td>↙</td><td></td></tr>
</table>

	(f,m)			WITH
	S		No surgeon cured every patient.	
	D	✓	$(\forall s)(\text{surgeon}(s) \rightarrow (\exists p)(\text{patient}(p) \rightarrow \neg \text{cure}(s, p)))$	
	I		$(\forall p)(\text{patient}(p) \rightarrow (\forall s)(\text{surgeon}(s) \rightarrow \neg \textit{cured by}(p, s)))$	
	r_1	↑	$r_1 = \frac{[1]}{[\text{all}_{\text{surgeon}}]}\lvert\text{surgeon}\rvert \leq \frac{[1, \ldots, \text{all}_{\text{patient}}]}{[1, \ldots, \text{few}_{\text{surgeon}}]}\lvert\text{surgeon}\rvert$	
	r_2	↙	$r_2 = \frac{[1, \ldots, \text{all}_{\text{surgeon}}]}{[\textit{all}_{\text{patient}}]}\lvert\text{patient}\rvert \leq \frac{[1, \ldots, \text{few}_{\text{surgeon}}]}{[1, \ldots, \text{all}_{\text{patient}}]}\lvert\text{patient}\rvert$	
	(m, m)			WITH
	S		No science student at university of Windsor attended every AI seminar of John McCarthy.	
	D	✓	$(\forall s)(\text{science student at university of Windsor}(s) \rightarrow (\exists r)(\text{AI seminar of John McCarthy}(r) \rightarrow \neg \text{attended}(s, r)))$	
	I	✓	$(\forall r)(\text{AI seminar of John McCarthy}(r) \rightarrow (\forall s)(\text{science student at university of Windsor}(s) \rightarrow \neg \text{attended by}(r, s)))$	
	r_1	↑	$r_1 = \frac{[1]}{[\text{all}_{\text{science student at university of Windsor}}]}\lvert\text{science student at university of Windsor}\rvert \leq \frac{[1, \ldots, \text{all}_{\text{seminar}}]}{[1, \ldots, \text{all}_{\text{student}}]}\lvert\text{student}\rvert$	
	r_2	↗	$r_2 = \frac{[1, \ldots, \text{all}_{\text{science student at university of Windsor}}]}{[\textit{all}_{\text{AI seminar of John McCarthy}}]}\lvert\text{AI seminar of John McCarthy}\rvert \leq \frac{[1, \ldots, \text{all}_{\text{student}}]}{[1, \ldots, \text{all}_{\text{seminar}}]}\lvert\text{seminar}\rvert$	
	(m, f)			WITH
	S		No MIT student read every book of Noam Chomsky.	
	D	✓	$(\forall s)(\text{MIT student}(s) \rightarrow (\exists b)(\text{book of Noam Chomsky}(b) \rightarrow \neg \text{read}(s, b)))$	
	I	✓	$(\forall b)(\text{book of Noam Chomsky}(b) \rightarrow (\forall s)(\text{MIT student}(s) \rightarrow \neg \text{read by}(b, s)))$	
	r_1	↑		

	r_2	↗	$r_1 = \frac{[1]}{[all_{\text{MIT student}}]} \lvert \text{MIT student} \rvert \leq \frac{[1,\ldots,few_{\text{book}}]}{[1,\ldots,all_{\text{student}}]} \lvert \text{student} \rvert$ $r_2 = \frac{[1,\ldots,all_{\text{MIT student}}]}{[all_{\text{book of Noam Chomsky}}]} \lvert \text{book of Noam Chomsky} \rvert \leq \frac{[1,\ldots,all_{\text{student}}]}{[1,\ldots,few_{\text{book}}]} \lvert \text{book} \rvert$	
	(f,1)		No PhD student is advised by every professor at U of W.	WITH
	S		$(\forall s)(\text{PhD student}(s) \rightarrow (\exists p)(\text{professor at U of W}(p) \rightarrow \neg \text{advised by}(s, p)))$	
	D	✓	$(\forall p)(\text{professor at U of W}(p) \wedge (\forall s)(\text{PhD student}(s) \rightarrow \neg \text{advise}(p, s)))$	
	I	✓	$r_1 = \frac{[1]}{[all_{\text{PhD student}}]} \lvert \text{PhD student} \rvert \leq \frac{[1]}{[1,\ldots,few_{\text{student}}]} \lvert \text{student} \rvert$	
	r_1	↑	$r_2 = \frac{[1,\ldots,all_{\text{PhD student}}]}{[all_{\text{professor at U of W}}]} \lvert \text{professor at U of W} \rvert \leq \frac{[1,\ldots,few_{\text{student}}]}{[1]} \lvert \text{professor} \rvert$	
	r_2	↗		
	(m, 1)		No software engineer is employed in every accounting company.	WITH
	S		$(\forall e)(\text{software engineer}(e) \rightarrow (\exists c)(\text{accounting company}(c) \rightarrow \neg \textit{employed}(e, c)))$	
	D	✓	$(\forall c)(\text{accounting company}(c) \wedge (\forall e)(\text{software engineer}(e) \rightarrow \neg \text{employed}(e, c)))$	
	I	✓	$r_1 = \frac{[1]}{[all_{\text{software engineer}}]} \lvert \text{software engineer} \rvert \leq \frac{[1]}{[1,\ldots,all_{\text{engineer}}]} \lvert \text{engineer} \rvert$	
	r_1	↑	$r_2 = \frac{[1,\ldots,all_{\text{software engineer}}]}{[all_{\text{accounting company}}]} \lvert \text{accounting company} \rvert \leq \frac{[1,\ldots,all_{\text{engineer}}]}{[1]} \lvert \text{company} \rvert$	
	r_2	↗		
	(1,1)		No graduate student is sitting on every chair in auditorium.	WITH
	S			

	D	✓	$(\forall s)(\text{graduate student}\ (s) \rightarrow (\exists c)(\text{chair in auditorium}\ (c) \rightarrow \neg \textit{sittingon}\ (s, j)))$	
	I	✓	$(\forall c)(\text{chair in auditorium}\ (c) \wedge (\forall s)(\text{graduate student}\ (s) \rightarrow \neg \text{sitting}\ (c, s)))$	
	r_1	↑	$r_1 = \frac{[1]}{[\text{all}_{\text{graduate student}}]} \lvert\text{graduate student}\rvert \leq \frac{[1]}{[1]} \lvert\text{student}\rvert$	
	r_2	↗	$r_2 = \frac{[1, \ldots, \text{all}_{\text{graduate student}}]}{[\textit{all}_{\text{chair in auditorium}}]} \lvert\text{chair in auditorium}\rvert \leq \frac{[1]}{[1]} \lvert\text{chair}\rvert$	

Table 3.2a: Results for sentences of the form $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ \text{R Every}\,[q_1, q_2, \ldots, q_n]C_2$ with examples.



No/Every	$[m_2=1]$	[Few]	[Many]
$[m_1=1]$	1	1	1
[Few]	1e		1
[Many]	1e	1e	1

Table 3.2b: Results for the example sentences of the form $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ \text{R Every}\,[q_1, q_2, \ldots, q_n]C_2$ are depicted in the table.

3.4.3 Type of Sentence $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ \text{R}\ \text{All}\,[q_1, q_2, \ldots, q_n]C_2$

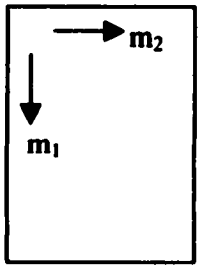
The results for sentences of the type $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ \text{R}\ \text{All}\,[q_1, q_2, \ldots, q_n]C_2$ are shown below. Sentences of this type with QC's mentioned in section 3.3.1 are considered:

No/All	(1,m)			WITH
	S		No university admits all prospective applicants.	
	D	✓	$(\forall u)(\text{university}(u) \rightarrow (\exists a)(\text{prospective applicant}\ (a) \rightarrow \neg \text{admit}\ (u, a)))$	
	I		$(\forall a)(\text{prospective applicant}\ (a) \rightarrow (\forall u)(\text{university}(u) \rightarrow \neg \text{admited by}\ (a, u)))$	
	r_1	↑	$r_1 = \frac{[1]}{[\text{all}_{\text{university}}]}\lvert\text{university}\rvert \leq \frac{[1,\ldots,\text{all}_{\text{applicant}}]}{[1]}\lvert\text{university}\rvert$	
	r_2	↙	$r_2 = \frac{[1,\ldots,\text{all}_{\text{university}}]}{[\text{all}_{\text{prospective applicant}}]}\lvert\text{prospective applicant}\rvert \leq \frac{[1]}{[1,\ldots,\text{all}_{\text{applicant}}]}\lvert\textit{applicant}\rvert$	
	(1, f)			WITH
	S		No professor is supervising all students.	
	D	✓	$(\forall p)(\text{professor}\,(p) \rightarrow (\exists s)(\text{student}\ (s) \rightarrow \neg \text{supervise}\ (p, s)))$	
	I		$(\forall s)(\text{student}\ (s) \rightarrow (\forall p)(\text{professor}\,(p) \rightarrow \neg \text{supervise by}\,(s, p)))$	
	r_1	↑	$r_1 = \frac{[1]}{[\text{all}_{\text{professor}}]}\lvert\text{professor}\rvert \leq \frac{[1,\ldots,\text{few}_{\text{student}}]}{[1]}\lvert\text{professor}\rvert$	
	r_2	↙	$r_2 = \frac{[1,\ldots,\text{all}_{\text{professor}}]}{[\text{all}_{\text{student}}]}\lvert\text{student}\rvert \leq \frac{[1]}{[1,\ldots\text{few}_{\text{student}}]}\lvert\text{student}\rvert$	

	(f,m)			WITH
	S		No MIT student submitted all papers to ACL.	
	D	✓	$(\forall s)(\text{MIT student}\ (s)\rightarrow(\exists p)(\text{paper to ACL}\ (p)\rightarrow\neg\text{submit}\ (s,p)))$	
	I		$(\forall p)(\text{paper to ACL}\ (p)\rightarrow(\forall s)(\text{MIT student}\ (s)\rightarrow\neg \textit{submitted}\ \text{by}\ (p,s)))$	
	r_1	↑	$r_1=\frac{[1]}{[\text{all}_{\text{MIT student}}]}\lvert\text{MIT student}\rvert\leq\frac{[1,\ldots,\text{all}_{\text{paper}}]}{[1,\ldots,\text{few}_{\text{student}}]}\lvert\text{student}\rvert$	
	r_2	↙	$r_2=\frac{[1,\ldots,\text{all}_{\text{MIT student}}]}{[\textit{all}_{\text{paper to ACL}}]}\lvert\text{paper to ACL}\rvert\leq\frac{[1,\ldots,\text{few}_{\text{student}}]}{[1,\ldots,\text{all}_{\text{paper}}]}\lvert\text{paper}\rvert$	
	(m, m)			WITH
	S		No science student at university of Windsor attended all AI seminar of John McCarthy.	
	D	✓	$(\forall s)(\text{science student at university of Windsor}\ (s)\rightarrow(\exists r)(\text{AI seminar of John McCarthy}\ (r)\rightarrow\neg\text{attended}\ (s,r)))$	
	I		$(\forall r)(\text{AI seminar of John McCarthy}\ (r)\rightarrow(\forall s)(\text{science student at university of Windsor}\ (s)\rightarrow\neg\text{attended by}\ (r,s)))$	
	r_1	↑	$r_1=\frac{[1]}{[\text{all}_{\text{science student at university of Windsor}}]}\lvert\text{science student at university of Windsor}\rvert\leq\frac{[1,\ldots,\text{all}_{\text{seminar}}]}{[1,\ldots,\text{all}_{\text{student}}]}\lvert\text{student}\rvert$	
	r_2	↗	$r_2=\frac{[1,\ldots,\text{all}_{\text{science student at university of Windsor}}]}{[\textit{all}_{\text{AI seminar of John McCarthy}}]}\lvert\text{AI seminar of John McCarthy}\rvert\leq\frac{[1,\ldots,\text{all}_{\text{student}}]}{[1,\ldots,\text{all}_{\text{seminar}}]}\lvert\text{seminar}\rvert$	
	(m, f)			WITH
	S		No Canadian reads all European newspapers.	
	D	✓	$(\forall p)(\text{person}\ (p)\rightarrow(\exists n)(\text{newspaper}\ (n)\rightarrow\neg\text{read}\ (c,p)))$	
	I		$(\forall n)(\text{newspaper}\ (n)\rightarrow(\forall p)(\text{person}\ (p)\rightarrow\neg\text{read by}\ (n,p)))$	
	r_1	↑	$r_1=\frac{[1]}{[\text{all}_{\text{Canadian person}}]}\lvert\text{Canadian person}\rvert\leq\frac{[1,\ldots,\text{few}_{\text{newspaper}}]}{[1,\ldots,\text{all}_{\text{person}}]}\lvert\text{person}\rvert$	
	r_2	↗	$r_2=\frac{[1,\ldots,\text{all}_{\text{Canadian person}}]}{[\textit{all}_{\text{European newspaper}}]}\lvert\text{European newspaper}\rvert\leq\frac{[1,\ldots,\text{all}_{\text{person}}]}{[1,\ldots,\text{few}_{\text{newspaper}}]}\lvert\text{newspaper}\rvert$	

	(f,1)		No PhD student is advised by all professors at university of Windsor.	WITH
	S		$(\forall s)(\text{PhD student}\ (s) \rightarrow (\exists p)(\text{professor at university of Windsor}\ (p) \rightarrow \neg \textit{advisedby}\ (s, p)))$	
	D	✓	$(\forall p)(\text{professor at university of Windsor}\ (p) \wedge (\forall s)(\text{PhD student}\ (s) \rightarrow \neg \text{advise}\ (p, s)))$	
	I	✓	$r_1 = \frac{[1]}{[\text{all}_{\text{PhD student}}]} \lvert \text{PhD student} \rvert \leq \frac{[1]}{[1, \ldots, \text{few}_{\text{student}}]} \lvert \text{student} \rvert$	
	r_1	↑	$r_2 = \frac{[1, \ldots, \text{all}_{\text{PhD student}}]}{[\textit{all}_{\text{professors at university of Windsor}}]} \lvert \text{professors at university of Windsor} \rvert \leq \frac{[1, \ldots, \text{few}_{\text{student}}]}{[1]} \lvert \text{professor} \rvert$	
	r_2	↗		
	(m, 1)		No software engineer is employed in all accounting companies.	WITH
	S		$(\forall e)(\text{software engineer}\ (e) \rightarrow (\exists c)(\text{accounting company}\ (c) \rightarrow \neg \textit{employed}\ (e, c)))$	
	D	✓	$(\forall c)(\text{accounting company}\ (c) \wedge (\forall e)(\text{software engineer}\ (e) \rightarrow \neg \text{employed}\ (e, c)))$	
	I	✓	$r_1 = \frac{[1]}{[\text{all}_{\text{software engineer}}]} \lvert \text{software engineer} \rvert \leq \frac{[1]}{[1, \ldots, \text{all}_{\text{engineer}}]} \lvert \text{engineer} \rvert$	
	r_1	↑	$r_2 = \frac{[1, \ldots, \text{all}_{\text{software engineer}}]}{[\textit{all}_{\text{accounting company}}]} \lvert \text{accounting company} \rvert \leq \frac{[1, \ldots, \text{all}_{\text{engineer}}]}{[1]} \lvert \text{company} \rvert$	
	r_2	↗		
	(1,1)		No MIT student is giving all AI seminars.	WITH
	S		$(\forall s)(\text{MIT student}\ (s) \rightarrow (\exists r)(\text{AI seminar}\ (j) \rightarrow \neg \textit{giving}\ (s, r)))$	
	D	✓	$(\forall r)(\text{AI seminar}\ (r) \wedge (\forall s)(\text{MIT student}\ (s) \rightarrow \neg \text{given by}\ (r, s)))$	
	I		$r_1 = \frac{[1]}{[\text{all}_{\text{MIT student}}]} \lvert \text{MIT student} \rvert \leq \frac{[1]}{[1]} \lvert \text{student} \rvert$	
	r_1	↑	$r_2 = \frac{[1, \ldots, \text{all}_{\text{MIT student}}]}{[\textit{all}_{\text{AI seminar}}]} \lvert \text{AI seminar} \rvert \leq \frac{[1]}{[1]} \lvert \text{seminar} \rvert$	
	r_2	↗		

Table 3.3a: Results for sentences of the form $\text{No}[p_1, p_2, \ldots, p_n]C_1 \text{ R All}[q_1, q_2, \ldots, q_n]C_2$ with examples.



No/All	[1]	[Few]	[Many]
[1]	1	1	1
[Few]	1e		1
[Many]	1e	1e	1

Table 3.3b: Results for the example sentences of the form $\text{No}[p_1, p_2, \ldots, p_n]C_1 \text{ R All}[q_1, q_2, \ldots, q_n]C_2$ are depicted in the table.

3.4.4 Type of Sentence $A\,[p_1, p_2, \ldots, p_n]C_1\ R\ no\,[q_1, q_2, \ldots, q_n]C_2$

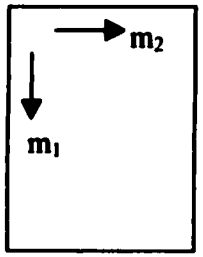
The results for sentences of the type $A\,[p_1, p_2, \ldots, p_n]C_1\ R\ no\,[q_1, q_2, \ldots, q_n]C_2$ are shown below. Sentences of this type with QC's mentioned in section 3.3.1 are considered:

A/No				
	(1,m)			WITH
	S		A graduate student at U of W won no NSERC award.	
	D	✓	$(\exists s)(\text{Graduate student of U of W}(s) \wedge (\forall a)(\text{NSERC award}(a) \rightarrow \neg\, \text{win}(s, a)))$	
	I		$(\forall a)(\text{NSERC award}(a) \rightarrow (\forall s)(\text{graduate student of U of W}(s) \rightarrow \neg\, \text{won By}(a, s)))$	
	r_1	↑	$r_1 = \frac{[1, \ldots, \text{all}_{\text{NSERC Award}}]}{[1]} \lvert \text{graduate student at U of W} \rvert \leq \frac{[1, \ldots, \text{all}_{\text{award}}]}{[1]} \lvert \textit{student} \rvert$	
	r_2	↙	$r_2 = \frac{[1, \ldots, \text{all}_{\text{graduate student at U of W}}]}{[\text{all}_{\text{NSERC award}}]} \lvert \text{NSERC award} \rvert \leq \frac{[1]}{[1, \ldots, \text{all}_{\text{award}}]} \lvert \textit{award} \rvert$	
	(1, f)			WITH
	S		A professor at U of W advises no PhD student.	
	D	✓	$(\exists p)(\text{professor at U of W}(p) \wedge (\forall s)(\text{PhD student}(s) \rightarrow \neg\, \text{advise}(p, s)))$	
	I		$(\forall s)(\text{PhD student}(s) \rightarrow (\forall p)(\text{professor at U of W}(p) \rightarrow \neg\, \text{advised by}(s, p)))$	
	r_1	↗	$r_1 = \frac{[1, \ldots, \text{all}_{\text{PhD student}}]}{[1]} \lvert \text{professor at U of W} \rvert \leq \frac{[1, \ldots, \text{few}_{\text{student}}]}{[1]} \lvert \text{professor} \rvert$	
	r_2	↑	$r_2 = \frac{[1, \ldots, \text{all}_{\text{professor at U of W}}]}{[\text{all}_{\text{PhD student}}]} \lvert \text{PhD student} \rvert \leq \frac{[1]}{[1, \ldots, \text{few}_{\text{student}}]} \lvert \text{student} \rvert$	

	(f,m)			REVERSE
	S		A MIT student submitted no paper on quantification.	
	D	✓	$(\exists s)(\text{MIT student}(s) \wedge (\forall p)(\text{paper on quantification}(p) \rightarrow \neg \text{submit}(s, p)))$	
	I		$(\forall p)(\text{paper on quantification}(p) \rightarrow (\forall s)(\text{MIT student}(s) \rightarrow \neg \text{submitted by}(p, s)))$	
	r_1	↙	$r_1 = \frac{[1,\ldots,\text{all}_{\text{paper on quantification}}]}{[1]} \lvert \text{MIT student} \rvert \leq \frac{[1,\ldots,\text{all}_{\text{paper}}]}{[1,\ldots,\text{few}_{\text{student}}]} \lvert \textit{student} \rvert$	
	r_2	↑	$r_2 = \frac{[1,\ldots,\text{all}_{\text{MIT student}}]}{[\text{all}_{\text{paper on quantifica tion}}]} \lvert \text{paper on quanrifica tion} \rvert \leq \frac{[1,\ldots,\text{few}_{\text{student}}]}{[1,\ldots,\text{all}_{\text{paper}}]} \lvert \textit{paper} \rvert$	
	(m, m)			REVERSE
	S		An American tourist visited no Indian restaurant in Washington.	
	D	✓	$(\exists t)(\text{American tourint}(t) \wedge (\forall r)(\text{Indian restaurant in Washington}(r) \rightarrow \neg \text{visited}(t, r)))$	
	I		$(\forall r)(\text{Indian restaurant in Washington}(r) \rightarrow (\forall t)(\text{American tourist}(t) \rightarrow \neg \text{visited by}(r, t)))$	
	r_1	↙	$r_1 = \frac{[1,\ldots,\text{all}_{\text{Indian restaurant in Washington}}]}{[1]} \lvert \text{American tourist} \rvert \leq \frac{[1,\ldots,\textit{all}_{\text{restaurant}}]}{[1,\ldots,\text{all}_{\text{tourist}}]} \lvert \textit{tourist} \rvert$	
	r_2	↑	$r_2 = \frac{[1,\ldots,\text{all}_{\text{American tourist}}]}{[\text{all}_{\text{Indian restaurant in Washington}}]} \lvert \text{Indian restaurant in Washington} \rvert \leq \frac{[1,\ldots,\text{all}_{\text{tourist}}]}{[1,\ldots,\text{all}_{\text{restaurant}}]} \lvert \text{restaurant} \rvert$	
	(m, f)			REVERSE
	S		A Canadian read no European newspaper.	
	D	✓	$(\exists a)(\text{Canadian person}(a) \rightarrow (\forall n)(\text{european news paper}(n) \rightarrow \neg \textit{read}(c, n)))$	
	I		$(\forall n)(\text{european newspaper}(n) \wedge (\forall c)(\text{Canadian person}(c) \rightarrow \neg \text{read by}(n, c)))$	
	r_1	↙	$r_1 = \frac{[1,\ldots,\text{all}_{\text{european newspaper}}]}{[1]} \lvert \text{Canadian person} \rvert \leq \frac{[1,\ldots,\textit{few}_{\textit{newspapers}}]}{[1,\ldots,\text{all}_{\text{person}}]} \lvert \textit{person} \rvert$	
	r_2	↑	$r_2 = \frac{[1,\ldots,\text{all}_{\text{canadian person}}]}{[\text{all}_{\text{european news paper}}]} \lvert \text{European newspaper} \rvert \leq \frac{[1,\ldots,\text{all}_{\text{person}}]}{[1,\ldots\text{few}_{\text{newspapers}}]} \lvert \textit{newspaper} \rvert$	

	(f,1) S D I r_1 r_2	 ✓ ↙ ↑	A faculty at university of Windsor is chairing no AI conference. $(\exists f)(\text{faculty at university of Windsor } (f) \wedge (\forall c)(\text{AI conference } (c) \rightarrow \neg \text{chairing } (f,c)))$ $(\forall c)(\text{AI conference}(c) \rightarrow (\forall f)(\text{faculty at university of Windsor } (f) \rightarrow \neg \text{chaired by } (c,f)))$ $r_1 = \frac{[1,\ldots,\text{all}_{\text{AI conference}}]}{[1]}\lvert\text{faculty at university of Windsor}\rvert \leq \frac{[1]}{[1,\ldots,\text{few}_{\text{faculty}}]}\lvert\text{faculty}\rvert$ $r_2 = \frac{[1,\ldots,\text{all}_{\text{faculty at university of Windsor}}]}{[\text{all}_{\text{AI conference}}]}\lvert\text{AI conference}\rvert \leq \frac{[1,\ldots,\text{few}_{\text{faculty}}]}{[1]}\lvert\text{conference}\rvert$	REVERSE
	(m, 1) S D I r_1 r_2	 ✓ ↙ ↑	A chemist works for no bank. $(\exists c)(\text{chemist}(c) \wedge (\forall b)(\text{bank } (b) \rightarrow \neg \text{works for}(c,b)))$ $(\forall b)(\text{bank}(b) \rightarrow (\forall c)(\text{chemist}(c) \rightarrow \neg \text{work for}(b,c)))$ $r_1 = \frac{[1,\ldots,\text{all}_{\text{bank}}]}{[1]}\lvert\text{chemist}\rvert \leq \frac{[1]}{[1,\ldots,\text{all}_{\text{chemist}}]}\lvert\text{chemist}\rvert$ $r_2 = \frac{[1,\ldots,\text{all}_{\text{chemist}}]}{[\text{all}_{\text{bank}}]}\lvert\text{bank}\rvert \leq \frac{[1,\ldots,\text{all}_{\text{chemist}}]}{[1]}\lvert\text{bank}\rvert$	REVERSE
	(1, 1) S D I r_1 r_2	 ✓ ↙ ↑	A student is giving no seminar. $(\exists s)(\text{student}(s) \rightarrow (\forall r)(\text{seminar}(r) \rightarrow \neg \text{giving } (s,r)))$ $(\forall r)(\text{seminar } (r) \wedge (\forall s)(\text{student}(s) \rightarrow \neg \text{given by } (r,s)))$ $r_1 = \frac{[1,\ldots,\text{all}_{\text{seminar}}]}{[1]}\lvert\text{student}\rvert \leq \frac{[1]}{[1]}\lvert\text{student}\rvert$ $r_2 = \frac{[1,\ldots,\text{all}_{\text{student}}]}{[\text{all}_{\text{seminar}}]}\lvert\text{seminar}\rvert \leq \frac{[1]}{[1]}\lvert\text{seminar}\rvert$	REVERSE

Table 3.4a Results for sentences of the form $A\,[p_1, p_2, \ldots, p_n]C_1\ R\ no\,[q_1, q_2, \ldots, q_n]C_2$ with examples.



A/No	[1]	[Few]	[Many]
[1]	0	1	1
[Few]	0		0
[Many]	0	0	0

Table 3.4b Results for the example sentences of the form $A\,[p_1, p_2, \ldots, p_n]C_1\ R\ no\,[q_1, q_2, \ldots, q_n]C_2$ are depicted in the table.

3.4.5 Type of Sentence $\text{Every}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{no}\,[q_1, q_2, \ldots, q_n]C_2$

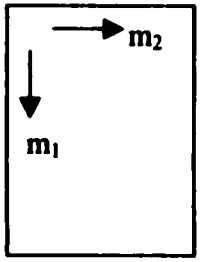
The results for sentences of the type $\text{Every}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{no}\,[q_1, q_2, \ldots, q_n]C_2$ are shown below. Sentences of this type with QC's mentioned in section 3.3.1 are considered:

Every/No	(1,m)		Every graduate student won no NSERC award. $(\forall s)(\text{Graduate student}\ (s) \wedge (\forall a)(\text{NSERC award}\,(a) \rightarrow \neg\, \text{win}(s, a)))$ $(\forall a)(\text{NSERC award}\,(a) \rightarrow (\exists s)(\text{graduate student}\ (s) \rightarrow \neg\, \text{won By}(a, s)))$ $r_1 = \frac{[1, \ldots, \text{all}_{\text{NSERC award}}]}{[\text{all}_{\text{graduate student}}]} \lvert\text{graduate student}\rvert \leq \frac{[1, \ldots, \text{all}_{\text{award}}]}{[1]} \lvert\text{student}\rvert$ $r_2 = \frac{[1]}{[\text{all}_{\text{NSERC award}}]} \lvert\text{NSERC award}\rvert \leq \frac{[1]}{[1, \ldots, \text{all}_{\text{award}}]} \lvert\text{award}\rvert$	WITH
	S			
	D	✓		
	I			
	r_1	↗		
	r_2	↑		
	(1, f)		Every professor at U of W advises no PhD student. $(\forall p)(\text{professor at U of W}\ (p) \wedge (\forall s)(\text{PhD student}\ (s) \rightarrow \neg \text{advise}\,(p, s)))$ $(\forall s)(\text{PhD student}\ (s) \rightarrow (\exists p)(\text{professor at U of W}\ (p) \rightarrow \neg\, \text{advised by}(s, p)))$ $r_1 = \frac{[1, \ldots, \text{all}_{\text{PhD student}}]}{[\text{all}_{\text{professor at U of W}}]} \lvert\text{professor at U of W}\rvert \leq \frac{[1, \ldots, \text{few}_{\text{student}}]}{[1]} \lvert\text{professor}\rvert$ $r_2 = \frac{[1]}{[\text{all}_{\text{PhD student}}]} \lvert\text{PhD student}\rvert \leq \frac{[1]}{[1, \ldots, \text{few}_{\text{student}}]} \lvert\text{student}\rvert$	WITH
	S			
	D	✓		
	I			
	r_1	↗		
	r_2	↑		

	(f,m)		Every MIT student submitted no paper on quantification. $(\forall s)(\text{MIT student}(s) \rightarrow (\forall p)(\text{paper on quantification}(p) \rightarrow \neg \text{submit}(s, p)))$ $(\forall p)(\text{paper on quantification}(p) \rightarrow (\exists s)(\text{MIT student}(s) \rightarrow \neg \text{submitted by}(p, s)))$ $r_1 = \frac{\lfloor 1, \ldots, \text{all}_{\text{paper on quantification}} \rfloor}{\lceil \text{all}_{\text{MIT student}} \rceil} \lvert \text{MIT student} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{paper}} \rfloor}{\lceil 1, \ldots, \text{few}_{\text{student}} \rceil} \lvert \textit{student} \rvert$ $r_2 = \frac{\lceil 1 \rceil}{\lceil \textit{all}_{\text{paper on quantification}} \rceil} \lvert \text{paper on quantification} \rvert \leq \frac{\lceil 1, \ldots, \text{few}_{\text{student}} \rceil}{\lceil 1, \ldots, \text{all}_{\text{paper}} \rceil} \lvert \text{paper} \rvert$	WITH
	S			
	D	✓		
	I			
	r_1	↗		
	r_2	↑		
	(m, m)		Every American tourist visited no Indian restaurant in Washington. $(\forall t)(\text{American tourist}(t) \rightarrow (\forall r)(\text{Indian restaurant in Washington}(r) \rightarrow \neg \text{visited}(t, r)))$ $(\forall r)(\text{Indian restaurant in Washington}(r) \rightarrow (\exists t)(\text{American tourist}(t) \rightarrow \neg \text{visited by}(r, t)))$ $r_1 = \frac{\lfloor 1, \ldots, \text{all}_{\text{Indian restaurant in Washington}} \rfloor}{\lceil \text{all}_{\text{American tourist}} \rceil} \lvert \text{American tourist} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{restaurant}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{tourist}} \rceil} \lvert \text{tourist} \rvert$ $r_2 = \frac{\lceil 1 \rceil}{\lceil \textit{all}_{\text{Indian restaurant in Washington}} \rceil} \lvert \text{Indian restaurant in Washington} \rvert \leq \frac{\lceil 1, \ldots, \text{all}_{\text{tourist}} \rceil}{\lceil 1, \ldots, \text{all}_{\text{museum}} \rceil} \lvert \text{restaurant} \rvert$	
	S			
	D	✓		
	I			
	r_1			
	r_2			
	(m, f)		Every one reads no newspaper. $(\forall p)(\text{person}(p) \rightarrow (\forall n)(\text{news paper}(n) \rightarrow \neg \text{read}(p, n)))$ $(\forall n)(\text{news paper}(n) \rightarrow (\exists p)(\text{person}(p) \rightarrow \neg \text{read by}(n, p)))$ $r_1 = \frac{\lfloor 1, .., \text{all}_{\text{newspaper}} \rfloor}{\lceil \text{all}_{\text{person}} \rceil} \lvert \text{person} \rvert \leq \frac{\lfloor 1, \ldots, \text{few}_{\text{news paper}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{person}} \rceil} \lvert \text{person} \rvert$ $r_2 = \frac{\lceil 1 \rceil}{\lceil \textit{all}_{\text{newspaper}} \rceil} \lvert \text{newspaper} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{person}} \rfloor}{\lceil 1, \ldots, \text{few}_{\text{news paper}} \rceil} \lvert \text{newspaper} \rvert$	REVERSE
	S			
	D	✓		
	I			
	r_1	↙		
	r_2	↑		

	(f,1)		Every faculty at U of W is chairing no conference. $(\forall f)(\text{faculty at U of W } (f) \rightarrow (\forall c)(\text{conference } (c) \rightarrow \neg \text{chairing } (f, c)))$ $(\forall c)(\text{conference } (c) \rightarrow (\exists f)(\text{faculty } (f) \rightarrow \neg \text{chaired by } (c, f)))$ $r_1 = \frac{[1,.., \text{all}_{\text{conference}}]}{[\text{all}_{\text{faculty at U of W}}]} \lvert \text{faculty at U of W} \rvert \leq \frac{[1]}{[1,..., \text{few}_{\text{faculty}}]} \lvert \text{faculty} \rvert$ $r_2 = \frac{[1]}{[\text{all}_{\text{conference}}]} \lvert \text{conference} \rvert \leq \frac{[1,..., \text{few}_{\text{faculty}}]}{[1]} \lvert \text{conference} \rvert$	REVERSE
	S			
	D	✓		
	I			
	r_1	↙		
	r_2	↑		
	(m, 1)		Every chemist works for no bank. $(\forall c)(\text{chemist } (c) \rightarrow (\forall b)(\text{bank } (b) \rightarrow \neg \text{works for } (c, b)))$ $(\forall b)(\text{bank } (b) \rightarrow (\exists c)(\text{chemist } (c) \rightarrow \neg \text{works for } (c, b)))$ $r_1 = \frac{[1,..., \text{all}_{\text{bank}}]}{[\text{all}_{\text{chemist}}]} \lvert \text{chemist} \rvert \leq \frac{[1]}{[1,..., \text{all}_{\text{chemist}}]} \lvert \text{chemist} \rvert$ $r_2 = \frac{[1]}{[\text{all}_{\text{bank}}]} \lvert \text{bank} \rvert \leq \frac{[1,..., \text{all}_{\text{chemist}}]}{[1]} \lvert \text{bank} \rvert$	REVERSE
	S			
	D	✓		
	I			
	r_1	↙		
	r_2	↑		
	(1,1)		Every student is giving no seminar. $(\forall s)(\text{student } (s) \rightarrow (\forall r)(\text{seminar } (r) \rightarrow \neg \textit{giving } (s, r)))$ $(\forall r)(\text{seminar } (r) \rightarrow (\exists s)(\text{student } (s) \rightarrow \neg \text{given by } (r, s)))$ $r_1 = \frac{[1,..., \text{all}_{\text{seminar}}]}{[\text{all}_{\text{student}}]} \lvert \text{student} \rvert \leq \frac{[1]}{[1]} \lvert \text{student} \rvert$ $r_2 = \frac{[1]}{[\text{all}_{\text{seminar}}]} \lvert \text{seminar} \rvert \leq \frac{[1]}{[1]} \lvert \text{seminar} \rvert$	WITH
	S			
	D	✓		
	I			
	r_1	↗		
	r_2	↑		

Table 3.5a: Results for sentences of the form Every $[p_1, p_2, \ldots, p_n]C_1$ R no $[q_1, q_2, \ldots, q_n]C_2$ with examples.



Every/No	[1]	[Few]	[Many]
[1]	1	1	1
[Few]	0		1
[Many]	0	0	1

Table 3.5b: Results for the example sentences of the form Every $[p_1, p_2, \ldots, p_n]C_1$ R no $[q_1, q_2, \ldots, q_n]C_2$ are depicted in the table.

3.4.6 Type of Sentence $\text{All}[p_1, p_2, \ldots, p_n]C_1 \text{ R no } [q_1, q_2, \ldots, q_n]C_2$

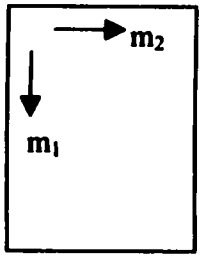
The results for sentences of the type $\text{All}[p_1, p_2, \ldots, p_n]C_1 \text{ R no } [q_1, q_2, \ldots, q_n]C_2$ are shown below. Sentences of this type with QC's mentioned in section 3.3.1 are considered:

All/No				
	(1,m)		All graduate students won no NSERC award.	WITH
	S		$(\forall s)(\text{Graduate student } (s) \wedge (\forall a)(\text{NSERC award } (a) \rightarrow \neg \text{ win}(s, a)))$	
	D	✓	$(\forall a)(\text{NSERC award } (a) \rightarrow (\exists s)(\text{graduate student } (s) \rightarrow \neg \text{ won By}(a, s)))$	
	I		$r_1 = \frac{[1, \ldots, \text{all}_{\text{NSERC award}}]}{[\text{all}_{\text{graduate student}}]} \lvert \text{graduate student} \rvert \leq \frac{[1, \ldots, \text{all}_{\text{award}}]}{[1]} \lvert \text{student} \rvert$	
	r_1	↗		
	r_2	↑	$r_2 = \frac{[1]}{[\text{all}_{\text{NSERC award}}]} \lvert \text{NSERC award} \rvert \leq \frac{[1]}{[1, \ldots, \text{all}_{\text{award}}]} \lvert \text{award} \rvert$	
	(1, f)		All professors at U of W advise no PhD student.	WITH
	S		$(\forall p)(\text{professor at U of W } (p) \wedge (\forall s)(\text{PhD student } (s) \rightarrow \neg \text{advise}(p, s)))$	
	D	✓	$(\forall s)(\text{PhD student } (s) \rightarrow (\exists p)(\text{professor at U of W } (p) \rightarrow \neg \text{ advised by}(s, p)))$	
	I		$r_1 = \frac{[1, \ldots, \text{all}_{\text{PhD student}}]}{[\text{all}_{\text{professor at U of W}}]} \lvert \text{professor at U of W} \rvert \leq \frac{[1, \ldots, \text{few}_{\text{student}}]}{[1]} \lvert \text{professor} \rvert$	
	r_1	↗		
	r_2	↑	$r_2 = \frac{[1]}{[\text{all}_{\text{PhD student}}]} \lvert \text{PhD student} \rvert \leq \frac{[1]}{[1, \ldots, \text{few}_{\text{student}}]} \lvert \text{student} \rvert$	

	(f,m)		All MIT students submitted no paper on quantification. $(\forall s)(\text{MIT student}(s) \rightarrow (\forall p)(\text{paper on quantification}(p) \rightarrow \neg \text{submit}(s, p)))$ $(\forall p)(\text{paper on quantification}(p) \rightarrow (\exists s)(\text{MIT student}(s) \rightarrow \neg \text{submitted by}(p, s)))$ $r_1 = \frac{\lfloor 1, \ldots, \text{all}_{\text{paper on quantification}} \rfloor}{\lceil \text{all}_{\text{MIT student}} \rceil} \lvert \text{MIT student} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{paper}} \rfloor}{\lceil 1, \ldots, \text{few}_{\text{student}} \rceil} \lvert \textit{student} \rvert$ $r_2 = \frac{\lceil 1 \rceil}{\lceil \textit{all}_{\text{paper on quantification}} \rceil} \lvert \text{paper on quantification} \rvert \leq \frac{\lceil 1, \ldots, \text{few}_{\text{student}} \rceil}{\lceil 1, \ldots, \text{all}_{\text{paper}} \rceil} \lvert \text{paper} \rvert$	WITH
	S			
	D	✓		
	I			
	r_1	↗		
	r_2	↑		
	(m, m)		All American tourists visited no Indian restaurant in Washington. $(\forall t)(\text{American tourist}(t) \rightarrow (\forall r)(\text{Indian restaurant in Washington}(r) \rightarrow \neg \text{visited}(t, r)))$ $(\forall r)(\text{Indian restaurant in Washington}(r) \rightarrow (\exists t)(\text{American tourist}(t) \rightarrow \neg \text{visited by}(r, t)))$ $r_1 = \frac{\lfloor 1, \ldots, \text{all}_{\text{Indian restaurant in Washington}} \rfloor}{\lceil \text{all}_{\text{American tourist}} \rceil} \lvert \text{American tourist} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{restaurant}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{tourist}} \rceil} \lvert \text{tourist} \rvert$ $r_2 = \frac{\lceil 1 \rceil}{\lceil \textit{all}_{\text{Indian restaurant in Washington}} \rceil} \lvert \text{Indian restaurant in Washington} \rvert \leq \frac{\lceil 1, \ldots, \text{all}_{\text{tourist}} \rceil}{\lceil 1, \ldots, \text{all}_{\text{museum}} \rceil} \lvert \text{restaurant} \rvert$	
	S			
	D	✓		
	I			
	r_1			
	r_2			
	(m, f)		All Canadian people read no European newspaper. $(\forall p)(\text{Canadian people}(p) \rightarrow (\forall n)(\text{European news paper}(n) \rightarrow \neg \text{read}(p, n)))$ $(\forall n)(\text{European news paper}(n) \rightarrow (\exists p)(\text{Canadian person}(p) \rightarrow \neg \text{read by}(n, p)))$ $r_1 = \frac{\lfloor 1, \ldots, \text{all}_{\text{European newspaper}} \rfloor}{\lceil \text{all}_{\text{Canadian person}} \rceil} \lvert \text{Canadian person} \rvert \leq \frac{\lfloor 1, \ldots, \text{few}_{\text{news paper}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{person}} \rceil} \lvert \text{person} \rvert$ $r_2 = \frac{\lceil 1 \rceil}{\lceil \textit{all}_{\text{European newspaper}} \rceil} \lvert \text{European newspaper} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{person}} \rfloor}{\lceil 1, \ldots, \text{few}_{\text{news paper}} \rceil} \lvert \text{newspaper} \rvert$	REVERSE
	S			
	D	✓		
	I			
	r_1	↙		
	r_2	↑		

	(f,1)		All professors at U of W are chairing no AI conference. $(\forall f)(\text{faculty at U of W } (f) \rightarrow (\forall c)(\text{AI conference } (c) \rightarrow \neg \text{chairing } (f, c)))$ $(\forall c)(\text{AI conference } (c) \rightarrow (\exists f)(\text{faculty } (f) \rightarrow \neg \text{chaired by } (c, f)))$	REVERSE
	S			
	D	✓		
	I			
	r_1	↙	$r_1 = \frac{[1,.., \text{all}_{\text{AI conference}}]}{[\text{all}_{\text{faculty at U of W}}]} \lvert\text{faculty at U of W}\rvert \leq \frac{[1]}{[1,..., \text{few}_{\text{faculty}}]} \lvert\text{faculty}\rvert$	
	r_2	↑	$r_2 = \frac{[1]}{[\text{all}_{\text{AI conference}}]} \lvert\text{AI conference}\rvert \leq \frac{[1,..., \text{few}_{\text{faculty}}]}{[1]} \lvert\text{conference}\rvert$	
	(m, 1)		All chemist works for no bank. $(\forall c)(\text{chemist } (c) \rightarrow (\forall b)(\text{bank } (b) \rightarrow \neg \text{works for } (c, b)))$ $(\forall b)(\text{bank } (b) \rightarrow (\exists c)(\text{chemist } (c) \rightarrow \neg \text{works for } (c, b)))$	REVERSE
	S			
	D	✓		
	I			
	r_1	↙	$r_1 = \frac{[1,..., \text{all}_{\text{bank}}]}{[\text{all}_{\text{chemist}}]} \lvert\text{chemist}\rvert \leq \frac{[1]}{[1,..., \text{all}_{\text{chemist}}]} \lvert\text{chemist}\rvert$	
	r_2	↑	$r_2 = \frac{[1]}{[\text{all}_{\text{bank}}]} \lvert\text{bank}\rvert \leq \frac{[1,..., \text{all}_{\text{chemist}}]}{[1]} \lvert\text{bank}\rvert$	
	(1,1)		All MIT students are giving no AI seminar. $(\forall s)(\text{MIT student } (s) \rightarrow (\forall r)(\text{AI seminar } (r) \rightarrow \neg \textit{giving } (s, r)))$ $(\forall r)(\text{AI seminar } (r) \rightarrow (\exists s)(\text{MIT student } (s) \rightarrow \neg \text{given by } (r, s)))$	WITH
	S			
	D	✓		
	I			
	r_1	↗	$r_1 = \frac{[1,..., \text{all}_{\text{AI seminar}}]}{[\text{all}_{\text{MIT student}}]} \lvert\text{MIT student}\rvert \leq \frac{[1]}{[1]} \lvert\text{student}\rvert$	
	r_2	↑	$r_2 = \frac{[1]}{[\text{all}_{\text{AI seminar}}]} \lvert\text{AI seminar}\rvert \leq \frac{[1]}{[1]} \lvert\text{seminar}\rvert$	

Table 3.6a: Results for sentences of the form $All[p_1, p_2,..., p_n]C_1$ R $no[q_1, q_2,..., q_n]C_2$ with examples.



All/No	[1]	[Few]	[Many]
[1]	1	1	1
[Few]	0		1
[Many]	0	0	1

Table 3.6b: Results for the example sentences of the form $All[p_1, p_2,..., p_n]C_1$ R $no[q_1, q_2,..., q_n]C_2$ are depicted in the table.

3.4.7 Type of Sentence $A\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$

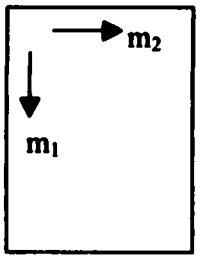
The results for sentences of the type $A\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$ are shown below. Sentences of this type with QC's mentioned in section 3.3.1 are considered:

A/Any	(1,m)			WITH
	S		A graduate student did not win any award.	
	D	✓	$(\exists s)(\text{Graduate student}\ (s) \wedge (\forall a)(\text{award}\,(a) \rightarrow \neg\, \text{win}(s, a)))$	
	I		$(\forall a)(\text{award}\,(a) \rightarrow (\exists s)(\text{graduate student}\ (s) \rightarrow \neg\, \text{won By}(a, s)))$	
	r_1	↑	$r_1 = \frac{[1, \ldots, \text{all}_{\text{Award}}]}{[1]} \lvert\text{graduate student}\rvert \leq \frac{[1, \ldots, \text{all}_{\text{award}}]}{[1]} \lvert \textit{student} \rvert$	
	r_2	↗	$r_2 = \frac{[1]}{[1, \ldots, \text{all}_{\text{award}}]} \lvert\text{award}\rvert \leq \frac{[1]}{[1, \ldots, \text{all}_{\text{award}}]} \lvert \textit{award} \rvert$	
	(1, f)		A professor at U of W does not advise any PhD student.	
	S		$(\exists p)(\text{professor at U of W}\,(p) \wedge (\forall s)(\text{PhD student}\,(s) \rightarrow \neg\, \text{advise}\,(p, s)))$	
	D	✓	$(\forall s)(\text{PhD student}\,(s) \rightarrow (\exists p)(\text{professor at U of W}\,(p) \rightarrow \neg\, \text{advised by}(s, p)))$	
	I		$r_1 = \frac{[1, \ldots, \text{all}_{\text{PhD student}}]}{[1]} \lvert\text{professor at U of W}\rvert \leq \frac{[1, \ldots, \text{few}_{\text{student}}]}{[1]} \lvert\text{professor}\rvert$	
	r_1		$r_2 = \frac{[1]}{[1, \ldots, \text{all}_{\text{PhD student}}]} \lvert\text{PhD student}\rvert \leq \frac{[1]}{[1, \ldots, \text{few}_{\text{student}}]} \lvert\text{student}\rvert$	
	r_2			

	(f,m)			REVERSE
	S		A MIT student did not submit any paper to ACL.	
	D	✓	$(\exists s)(\text{MIT student}(s) \wedge (\forall p)(\text{paper to ACL}(p) \rightarrow \neg \text{submit}(s, p)))$	
	I		$(\forall p)(\text{paper to ACL}(p) \rightarrow (\exists s)(\text{MIT student}(s) \rightarrow \neg \text{submitted by}(p, s)))$	
	r_1	↙	$r_1 = \frac{[1,\ldots,\text{all}_{\text{paper to ACL}}]}{[1]} \lvert \text{MIT student} \rvert \leq \frac{[1,\ldots,\text{all}_{\text{paper}}]}{[1,\ldots,\text{few}_{\text{student}}]} \lvert \textit{student} \rvert$	
	r_2	↑	$r_2 = \frac{[1]}{[1,\ldots,\text{all}_{\text{paper to ACL}}]} \lvert \text{paper to ACL} \rvert \leq \frac{[1,\ldots,\text{few}_{\text{student}}]}{[1,\ldots,\text{all}_{\text{paper}}]} \lvert \textit{paper} \rvert$	
	(m, m)			REVERSE
	S		A MIT student did not attend any AI seminar.	
	D	✓	$(\exists s)(\text{MIT student}(s) \wedge (\forall r)(\text{AI seminar}(r) \rightarrow \neg \text{attended}(s, r)))$	
	I		$(\forall r)(\text{AI seminar}(r) \rightarrow (\exists s)(\text{MIT student}(s) \rightarrow \neg \text{attended by}(r, s)))$	
	r_1	↙	$r_1 = \frac{[1,\ldots,\text{all}_{\text{AI seminar}}]}{[1]} \lvert \text{MIT student} \rvert \leq \frac{[1,\ldots,\textit{all}_{\text{seminar}}]}{[1,\ldots,\text{all}_{\text{student}}]} \lvert \text{student} \rvert$	
	r_2	↑	$r_2 = \frac{[1]}{[1,\ldots,\text{all}_{\text{AI seminar}}]} \lvert \text{AI seminar} \rvert \leq \frac{[1,\ldots,\text{all}_{\text{student}}]}{[1,\ldots,\text{all}_{\text{seminar}}]} \lvert \text{seminar} \rvert$	
	(m, f)			REVERSE
	S		A Canadian did not read any European newspaper.	
	D	✓	$(\exists p)(\text{Canadian person}(p) \rightarrow (\forall n)(\text{european news paper}(n) \rightarrow \neg \textit{read}(c, n)))$	
	I		$(\forall n)(\text{european newspaper}(n) \wedge (\exists p)(\text{Canadian person}(p) \rightarrow \neg \text{read by}(n, p)))$	
	r_1	↙	$r_1 = \frac{[1,\ldots,\text{all}_{\text{european newspaper}}]}{[1]} \lvert \text{Canadian person} \rvert \leq \frac{[1,\ldots,\textit{few}_{\textit{newspapers}}]}{[1,\ldots,\text{all}_{\text{person}}]} \lvert \textit{person} \rvert$	
	r_2	↑	$r_2 = \frac{[1]}{[1,\ldots,\text{all}_{\text{european news paper}}]} \lvert \text{European newspaper} \rvert \leq \frac{[1,\ldots,\text{all}_{\text{person}}]}{[1,\ldots\text{few}_{\text{newspapers}}]} \lvert \textit{newspaper} \rvert$	

	(f,1)		A faculty at U of W is not chairing any conference.	REVERSE
	S		$(\exists f)(\text{faculty at U of W}(f) \rightarrow (\forall c)(\text{conference}(c) \rightarrow \neg \textit{chair}(f, c)))$	
	D	✓	$(\forall c)(\text{conference}(c) \wedge (\exists f)(\text{faculty at U of W}(f) \rightarrow \neg \text{chaired by}(c, f)))$	
	I		$r_1 = \frac{[1,\ldots,\text{all}_{\text{conference}}]}{[1]}\lvert\text{faculty at U of W}\rvert \leq \frac{[1]}{[1,\ldots,\textit{few}_{\text{faculty}}]}\lvert\text{faculty}\rvert$	
	r_1	↙		
	r_2	↑	$r_2 = \frac{[1]}{[1,\ldots,\text{all}_{\text{conference}}]}\lvert\text{conference}\rvert \leq \frac{[1,\ldots,\text{few}_{\text{faculty}}]}{[1]}\lvert\text{conference}\rvert$	
	(m, 1)		A chemist does not work for any bank.	REVERSE
	S		$(\exists c)(\text{chemist}(c) \wedge (\forall b)(\text{bank}(b) \rightarrow \neg \text{works for}(c, b)))$	
	D	✓	$(\forall b)(\text{bank}(b) \rightarrow (\exists c)(\text{chemist}(c) \rightarrow \neg \text{work for}(c, b)))$	
	I		$r_1 = \frac{[1,\ldots,\text{all}_{\text{bank}}]}{[1]}\lvert\text{chemist}\rvert \leq \frac{[1]}{[1,\ldots,\text{all}_{\text{chemist}}]}\lvert\text{chemist}\rvert$	
	r_1	↙		
	r_2	↑	$r_2 = \frac{[1]}{[1,\ldots,\text{all}_{\text{bank}}]}\lvert\text{bank}\rvert \leq \frac{[1,\ldots,\text{all}_{\text{chemist}}]}{[1]}\lvert\text{bank}\rvert$	
	(1, 1)		A student is not giving any seminar.	REVERSE
	S		$(\exists s)(\text{student}(s) \wedge (\forall r)(\text{seminar}(r) \rightarrow \neg \text{giving}(s, r)))$	
	D	✓	$(\forall r)(\text{seminar}(r) \rightarrow (\exists s)(\text{student}(s) \rightarrow \neg \text{given by}(r, s)))$	
	I		$r_1 = \frac{[1,\ldots,\text{all}_{\text{seminar}}]}{[1]}\lvert\text{seminar}\rvert \leq \frac{[1]}{[1]}\lvert\text{seminar}\rvert$	
	r_1	↙		
	r_2	↑	$r_2 = \frac{[1]}{[1,\ldots,\text{all}_{\text{seminar}}]}\lvert\text{seminar}\rvert \leq \frac{[1]}{[1]}\lvert\text{seminar}\rvert$	

Table 3.7a: Results for sentences of the form $A\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$ with examples.



A/Any	[1]	[Few]	[Many]
[1]	0	0	1
[Few]	0		0
[Many]	0	0	0

Table 3.7b: Results for the example sentences of the form $A\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$ are depicted in the table.

3.4.8 Type of Sentence $A\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{all}\,[q_1, q_2, \ldots, q_n]C_2$

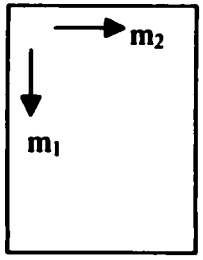
The results for sentences of the type $A\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{all}\,[q_1, q_2, \ldots, q_n]C_2$ are shown below. Sentences of this type with QC's mentioned in section 3.3.1 are considered:

A/All	(1,m)			REVERSE
	S		A student did not win all awards.	
	D	✓	$(\exists s)(\text{student}(s) \wedge (\forall a)(\text{award}(a) \rightarrow \neg\, \text{win}(s, a)))$	
	I		$(\forall a)(\text{award}(a) \rightarrow (\exists s)(\text{student}(s) \rightarrow \neg\, \text{won By}(a, s)))$	
	r_1	↙	$r_1 = \frac{[\text{all}_{\text{Award}}]}{[1]}\lvert\text{student}\rvert \leq \frac{[1,\ldots,\text{all}_{\text{award}}]}{[1]}\lvert\text{student}\rvert$	
	r_2	↑	$r_2 = \frac{[1]}{[\text{all}_{\text{award}}]}\lvert\text{award}\rvert \leq \frac{[1]}{[1,\ldots,\text{all}_{\text{award}}]}\lvert\text{award}\rvert$	
	(1, f)			REVERSE
	S		A professor at U of W does not advise all students.	
	D	✓	$(\exists p)(\text{professor}(p) \wedge (\forall s)(\text{student}(s) \rightarrow \neg\, \text{advise}(p, s)))$	
	I		$(\forall s)(\text{student}(s) \rightarrow (\exists p)(\text{professor}(p) \rightarrow \neg\, \text{advised by}(s, p)))$	
	r_1	↙	$r_1 = \frac{[\text{all}_{\text{student}}]}{[1]}\lvert\text{professor}\rvert \leq \frac{[1,\ldots,\text{few}_{\text{student}}]}{[1]}\lvert\text{professor}\rvert$	
	r_2	↑	$r_2 = \frac{[1]}{[1,\ldots,\text{all}_{\text{student}}]}\lvert\text{student}\rvert \leq \frac{[1]}{[1,\ldots,\text{few}_{\text{student}}]}\lvert\text{student}\rvert$	

	(f,m)		A MIT student did not submit all paper to ACL. $(\exists s)(\text{MIT student}(s) \wedge (\forall p)(\text{paper to ACL}(p) \rightarrow \neg \text{submit}(s, p)))$ $(\forall p)(\text{paper to ACL}(p) \rightarrow (\exists s)(\text{MIT student}(s) \rightarrow \neg \text{submitted by}(p, s)))$ $r_1 = \frac{\lfloor \text{all}_{\text{paper to ACL}} \rfloor}{\lceil 1 \rceil} \lvert \text{MIT student} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{paper}} \rfloor}{\lceil 1, \ldots, \text{few}_{\text{student}} \rceil} \lvert \textit{student} \rvert$ $r_2 = \frac{\lceil 1 \rceil}{\lceil \text{all}_{\text{paper to ACL}} \rceil} \lvert \text{paper to ACL} \rvert \leq \frac{\lceil 1, \ldots, \text{few}_{\text{student}} \rceil}{\lceil 1, \ldots, \text{all}_{\text{paper}} \rceil} \lvert \textit{paper} \rvert$	REVERSE
	S			
	D	✓		
	I			
	r_1	↙		
	r_2	↑		
	(m, m)		A MIT student did not attend all AI seminars. $(\exists s)(\text{MIT student}(s) \wedge (\forall r)(\text{AI seminar}(r) \rightarrow \neg \text{attended}(s, r)))$ $(\forall r)(\text{AI seminar}(r) \rightarrow (\exists s)(\text{MIT student}(s) \rightarrow \neg \text{attended by}(r, s)))$ $r_1 = \frac{\lfloor \text{all}_{\text{AI seminar}} \rfloor}{\lceil 1 \rceil} \lvert \text{MIT student} \rvert \leq \frac{\lfloor 1, \ldots, \textit{all}_{\text{seminar}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{student}} \rceil} \lvert \text{student} \rvert$ $r_2 = \frac{\lceil 1 \rceil}{\lceil \text{all}_{\text{AI seminar}} \rceil} \lvert \text{AI seminar} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{student}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{seminar}} \rceil} \lvert \text{seminar} \rvert$	REVERSE
	S			
	D	✓		
	I			
	r_1	↙		
	r_2	↑		
	(m, f)		A Canadian did not read all European newspaper. $(\exists p)(\text{Canadian person}(p) \rightarrow (\forall n)(\text{european news paper}(n) \rightarrow \neg \textit{read}(c, n)))$ $(\forall n)(\text{european newspaper}(n) \wedge (\exists p)(\text{Canadian person}(p) \rightarrow \neg \text{read by}(n, p)))$ $r_1 = \frac{\lfloor \text{all}_{\text{european newspaper}} \rfloor}{\lceil 1 \rceil} \lvert \text{Canadian person} \rvert \leq \frac{\lfloor 1, \ldots, \textit{few}_{\textit{newspapers}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{person}} \rceil} \lvert \textit{person} \rvert$ $r_2 = \frac{\lceil 1 \rceil}{\lceil \text{all}_{\text{european news paper}} \rceil} \lvert \text{European newspaper} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{person}} \rfloor}{\lceil 1, \ldots \text{few}_{\text{newspapers}} \rceil} \lvert \textit{newspaper} \rvert$	REVERSE
	S			
	D	✓		
	I			
	r_1	↙		
	r_2	↑		

	(f,1)		A faculty at U of W is not chairing all conference.	REVERSE
	S		$(\exists f)(\text{faculty at U of W}(f) \rightarrow (\forall c)(\text{conference}(c) \rightarrow \neg \textit{chair}(f, c)))$	
	D	✓	$(\forall c)(\text{conference}(c) \wedge (\exists f)(\text{faculty at U of W}(f) \rightarrow \neg \text{chaired by}(c, f)))$	
	I		$r_1 = \frac{\lceil \text{all}_{\text{conference}} \rceil}{\lceil 1 \rceil} \lvert \text{faculty at U of W} \rvert \leq \frac{\lceil 1 \rceil}{\lceil 1, \ldots, \textit{few}_{\text{faculty}} \rceil} \lvert \text{faculty} \rvert$	
	r_1	↙		
	r_2	↑	$r_2 = \frac{\lceil 1 \rceil}{\lceil \text{all}_{\text{conference}} \rceil} \lvert \text{conference} \rvert \leq \frac{\lceil 1, \ldots, \text{few}_{\text{faculty}} \rceil}{\lceil 1 \rceil} \lvert \text{conference} \rvert$	
	(m, 1)		An engineer does not work for all companies.	REVERSE
	S		$(\exists e)(\text{engineer}(e) \wedge (\forall c)(\text{company}(c) \rightarrow \neg \text{works for}(e, c)))$	
	D	✓	$(\forall c)(\text{company}(c) \rightarrow (\exists e)(\text{engineer}(e) \rightarrow \neg \text{work for}(e, c)))$	
	I		$r_1 = \frac{\lceil \text{all}_{\text{companies}} \rceil}{\lceil 1 \rceil} \lvert \text{engineer} \rvert \leq \frac{\lceil 1 \rceil}{\lceil 1, \ldots, \text{all}_{\text{engineer}} \rceil} \lvert \text{engineer} \rvert$	
	r_1	↙		
	r_2	↑	$r_2 = \frac{\lceil 1 \rceil}{\lceil \text{all}_{\text{company}} \rceil} \lvert \text{company} \rvert \leq \frac{\lceil 1, \ldots, \text{all}_{\text{engineer}} \rceil}{\lceil 1 \rceil} \lvert \text{company} \rvert$	
	(1, 1)		A student is not giving all seminars.	REVERSE
	S		$(\exists s)(\text{student}(s) \wedge (\forall r)(\text{seminar}(r) \rightarrow \neg \text{giving}(s, r)))$	
	D	✓	$(\forall r)(\text{seminar}(r) \rightarrow (\exists s)(\text{student}(s) \rightarrow \neg \text{given by}(r, s)))$	
	I		$r_1 = \frac{\lceil \text{all}_{\text{seminar}} \rceil}{\lceil 1 \rceil} \lvert \text{seminar} \rvert \leq \frac{\lceil 1 \rceil}{\lceil 1 \rceil} \lvert \text{seminar} \rvert$	
	r_1	↙		
	r_2	↑	$r_2 = \frac{\lceil 1 \rceil}{\lceil \text{all}_{\text{seminar}} \rceil} \lvert \text{seminar} \rvert \leq \frac{\lceil 1 \rceil}{\lceil 1 \rceil} \lvert \text{seminar} \rvert$	

Table 3.8a: Results for sentences of the form $A\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ all\,[q_1, q_2, \ldots, q_n]C_2$ with examples.



A/All	[1]	[Few]	[Many]
[1]	0	0	0
[Few]	0	0	0
[Many]	0	0	0

Table 3.8b: Results for the example sentences of the form $A\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ all\,[q_1, q_2, \ldots, q_n]C_2$ are depicted in the table.

3.4.9 Type of Sentence $\text{Every}\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$

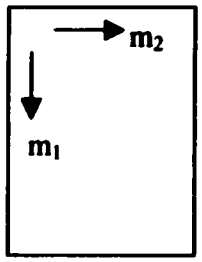
The results for sentences of the type $\text{Every}\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$ are shown below. Sentences of this type with QC's mentioned in section 3.3.1 are considered:

Every/Any	(1,m)			WITH
	S		Every student did not win any award.	
	D	✓	$(\forall s)(\text{student}\ (s) \wedge (\forall a)(\text{award}\,(a) \rightarrow \neg\, \text{win}(s, a)))$	
	I		$(\forall a)(\text{award}\,(a) \rightarrow (\forall s)(\text{student}\ (s) \rightarrow \neg\, \text{won By}(a, s)))$	
	r_1	↑	$r_1 = \frac{[1, \ldots, \text{all}_{\text{award}}]}{[\text{all}_{\text{student}}]} \lvert\text{student}\rvert \leq \frac{[1, \ldots, \text{all}_{\text{award}}]}{[1]} \lvert\text{student}\rvert$	
	r_2	↙	$r_2 = \frac{[\text{all}_{\text{student}}]}{[1, \ldots, \text{all}_{\text{award}}]} \lvert\text{award}\rvert \leq \frac{[1]}{[1, \ldots, \text{all}_{\text{award}}]} \lvert\text{award}\rvert$	
	(1, f)			WITH
	S		Every MIT professor does not advise any PhD student.	
	D	✓	$(\forall p)(\text{MIT professor}\ (p) \wedge (\forall s)(\text{PhD student}\ (s) \rightarrow \neg\, \text{advise}(p, s)))$	
	I		$(\forall s)(\text{PhD student}\ (s) \rightarrow (\forall p)(\text{MIT professor}\ (p) \rightarrow \neg\, \text{advised by}(s, p)))$	
	r_1	↑	$r_1 = \frac{[1, \ldots, \text{all}_{\text{PhD student}}]}{[\text{all}_{\text{MIT professor}}]} \lvert\text{MIT professor}\rvert \leq \frac{[1, \ldots, \text{few}_{\text{student}}]}{[1]} \lvert\text{professor}\rvert$	
	r_2	↙	$r_2 = \frac{[\text{all}_{\text{MIT professor}}]}{[1, \ldots, \text{all}_{\text{PhD student}}]} \lvert\text{PhD student}\rvert \leq \frac{[1]}{[1, \ldots, \text{few}_{\text{student}}]} \lvert\text{student}\rvert$	

	(f,m)		Every MIT student did not submit any paper on quantification. $(\forall s)(\text{MIT student}(s) \rightarrow (\forall p)(\text{paper on quantification}(p) \rightarrow \neg \text{submit}(s, p)))$ $(\forall p)(\text{paper on quantification}(p) \rightarrow (\forall s)(\text{MIT student}(s) \rightarrow \neg \text{submitted by}(p, s)))$ $r_1 = \frac{\lfloor 1, \ldots, \text{all}_{\text{paper on quantification}} \rfloor}{\lceil \text{all}_{\text{MIT student}} \rceil} \lvert \text{MIT student} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{paper}} \rfloor}{\lceil 1, \ldots, \text{few}_{\text{student}} \rceil} \lvert \textit{student} \rvert$ $r_2 = \frac{\lceil \text{all}_{\text{MIT student}} \rceil}{\lceil 1, \ldots, \textit{all}_{\text{paper on quantification}} \rceil} \lvert \text{paper on quantification} \rvert \leq \frac{\lceil 1, \ldots, \text{few}_{\text{student}} \rceil}{\lceil 1, \ldots, \text{all}_{\text{paper}} \rceil} \lvert \text{paper} \rvert$	WITH
	S			
	D	✓		
	I			
	r_1	↑		
	r_2	↙		
	(m, m)		Every MIT student did not attend any seminar of Dr. Millar. $(\forall s)(\text{MIT student}(s) \rightarrow (\forall r)(\text{seminar of Dr. Millar}(r) \rightarrow \neg \text{attended}(s, r)))$ $(\forall r)(\text{seminar of Dr. Millar}(r) \rightarrow (\forall s)(\text{MIT student}(s) \rightarrow \neg \text{attended by}(r, s)))$ $r_1 = \frac{\lfloor 1, \ldots, \text{all}_{\text{seminar of Dr. Millar}} \rfloor}{\lceil \text{all}_{\text{MIT student}} \rceil} \lvert \text{MIT student} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{seminar}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{student}} \rceil} \lvert \text{student} \rvert$ $r_2 = \frac{\lceil \text{all}_{\text{MIT student}} \rceil}{\lceil 1, \ldots, \textit{all}_{\text{seminar of Dr. Millar}} \rceil} \lvert \text{seminar of Dr. Millar} \rvert \leq \frac{\lceil 1, \ldots, \text{all}_{\text{student}} \rceil}{\lceil 1, \ldots, \text{all}_{\text{seminar}} \rceil} \lvert \text{seminar} \rvert$	WITH
	S			
	D	✓		
	I			
	r_1	↑		
	r_2	↙		
	(m, f)		Every one did not read any newspaper. $(\forall p)(\text{person}(p) \rightarrow (\forall n)(\text{news paper}(n) \rightarrow \neg \text{read}(p, n)))$ $(\forall n)(\text{news paper}(n) \rightarrow (\forall p)(\text{person}(p) \rightarrow \neg \text{read by}(n, p)))$ $r_1 = \frac{\lfloor 1, \ldots, \text{all}_{\text{newspaper}} \rfloor}{\lceil \text{all}_{\text{person}} \rceil} \lvert \text{person} \rvert \leq \frac{\lfloor 1, \ldots, \text{few}_{\text{news paper}} \rfloor}{\lceil 1, \ldots, \text{all}_{\text{person}} \rceil} \lvert \text{person} \rvert$ $r_2 = \frac{\lfloor \text{all}_{\text{person}} \rfloor}{\lceil 1, \ldots, \textit{all}_{\text{newspaper}} \rceil} \lvert \text{newspaper} \rvert \leq \frac{\lfloor 1, \ldots, \text{all}_{\text{person}} \rfloor}{\lceil 1, \ldots, \text{few}_{\text{news paper}} \rceil} \lvert \text{newspaper} \rvert$	REVERSE
	S			
	D	✓		
	I			
	r_1	↙		
	r_2	↑		

	(f,1) S D I r_1 r_2	 ✓ ↙ ↑	Every faculty at U of W is not chairing any conference. $(\forall f)(\text{faculty at U of W } (f) \rightarrow (\forall c)(\text{conference } (c) \rightarrow \neg \text{chairing } (f, c)))$ $(\forall c)(\text{conference } (c) \rightarrow (\forall f)(\text{faculty } (f) \rightarrow \neg \text{chaired by } (c, f)))$ $r_1 = \frac{[1,..,\text{all}_{\text{conference}}]}{[\text{all}_{\text{faculty at U of W}}]} \lvert\text{faculty at U of W}\rvert \leq \frac{[1]}{[1,...,\text{few}_{\text{faculty}}]} \lvert\text{faculty}\rvert$ $r_2 = \frac{[\text{all}_{\text{faculty at U of W}}]}{[1,...,\mathit{all}_{\text{conference}}]} \lvert\text{conference}\rvert \leq \frac{[1,...,\text{few}_{\text{faculty}}]}{[1]} \lvert\text{conference}\rvert$	REVERSE
	(m, 1) S D I r_1 r_2	 ✓ ↙ ↑	Every software engineer does not work for any accounting company. $(\forall e)(\text{software engineer } (e) \rightarrow (\forall c)(\text{accounting company } (c) \rightarrow \neg \text{works for } (e, c)))$ $(\forall c)(\text{accounting company } (c) \rightarrow (\forall e)(\text{software engineer } (e) \rightarrow \neg \text{works for } (e, c)))$ $r_1 = \frac{[1,...,\text{all}_{\text{accounting company}}]}{[\text{all}_{\text{software engineer}}]} \lvert\text{software engineer}\rvert \leq \frac{[1]}{[1,...,\text{all}_{\text{engineer}}]} \lvert\text{engineer}\rvert$ $r_2 = \frac{[\text{all}_{\text{software engineer}}]}{[1,...,\mathit{all}_{\text{accounting company}}]} \lvert\text{accounting company}\rvert \leq \frac{[1,...,\text{all}_{\text{engineer}}]}{[1]} \lvert\text{company}\rvert$	REVERSE
	(1,1) S D I r_1 r_2	 ✓ ↑ ↙	Every student is not giving any seminar. $(\forall s)(\text{student } (s) \rightarrow (\forall r)(\text{seminar } (r) \rightarrow \neg \mathit{giving } (s, r)))$ $(\forall r)(\text{seminar } (r) \rightarrow (\forall s)(\text{student } (s) \rightarrow \neg \text{given by } (r, s)))$ $r_1 = \frac{[1,...,\text{all}_{\text{seminar}}]}{[\text{all}_{\text{student}}]} \lvert\text{student}\rvert \leq \frac{[1]}{[1]} \lvert\text{student}\rvert$ $r_2 = \frac{[\text{all}_{\text{student}}]}{[1,...,\mathit{all}_{\text{seminar}}]} \lvert\text{seminar}\rvert \leq \frac{[1]}{[1]} \lvert\text{seminar}\rvert$	WITH

Table 3.9a: Results for sentences of the form Every $[p_1, p_2, \ldots, p_n]C_1$ $\neg$R any $[q_1, q_2, \ldots, q_n]C_2$ with examples.



Every/Any	[1]	[Few]	[Many]
[1]	1	1	1
[Few]	0		1
[Many]	0	0	1

Table 3.9b: Results for the example sentences of the form Every $[p_1, p_2, \ldots, p_n]C_1$ $\neg$R any $[q_1, q_2, \ldots, q_n]C_2$ are depicted in the table.

3.4.10 Type of Sentence $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$

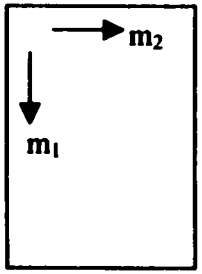
The results for sentences of the type $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$ are shown below. Sentences of this type with QC's mentioned in section 3.3.1 are considered:

No/Any	(1,m)		No student won any award. $(\forall s)(\text{student}(s) \wedge (\exists a)(\text{award}(a) \rightarrow \neg \text{win}(s, a)))$ $(\forall a)(\text{award}(a) \rightarrow (\forall s)(\text{student}(s) \rightarrow \neg \text{won By}(a, s)))$ $r_1 = \frac{[1]}{[\text{all}_{\text{student}}]}\lvert\text{student}\rvert \leq \frac{[1, \ldots, \text{all}_{\text{award}}]}{[1]}\lvert\text{student}\rvert$ $r_2 = \frac{[1, \ldots, \text{all}_{\text{student}}]}{[\text{all}_{\text{award}}]}\lvert\text{award}\rvert \leq \frac{[1]}{[1, \ldots, \text{all}_{\text{award}}]}\lvert\text{award}\rvert$	REVERSE
	S			
	D			
	I	✓		
	r_1	↑		
	r_2	↙		
	(1, f)		No professor at U of W advises any PhD student. $(\forall p)(\text{professor at U of W}(p) \rightarrow (\exists s)(\text{PhD student}(s) \rightarrow \neg \text{advise}(p, s)))$ $(\forall s)(\text{PhD student}(s) \rightarrow (\forall p)(\text{professor at U of W}(p) \rightarrow \neg \text{advised by}(s, p)))$ $r_1 = \frac{[1]}{[\text{all}_{\text{professor at U of W}}]}\lvert\text{professor at U of W}\rvert \leq \frac{[1, \ldots, \text{few}_{\text{student}}]}{[1]}\lvert\text{professor}\rvert$ $r_2 = \frac{[1, \ldots, \text{all}_{\text{professor at U of W}}]}{[\text{all}_{\text{PhD student}}]}\lvert\text{PhD student}\rvert \leq \frac{[1]}{[1, \ldots \text{few}_{\text{student}}]}\lvert\text{student}\rvert$	REVERSE
	S			
	D			
	I	✓		
	r_1	↑		
	r_2	↙		

	(f,m)		No MIT student submitted any paper to ACL. $(\forall s)(\text{MIT student}\ (s)\rightarrow(\exists p)(\text{paper to ACL}\ (p)\rightarrow\neg\text{submitted}\ (s,p)))$ $(\forall p)(\text{paper to ACL}\ (p)\rightarrow(\forall s)(\text{MIT student}\ (s)\rightarrow\neg\textit{submitted}\ \text{by}\ (p,s)))$ $r_1=\frac{[1]}{[\text{all}_{\text{MIT student}}]}\lvert\text{MIT student}\rvert\leq\frac{[1,\ldots,\text{all}_{\text{paper}}]}{[1,\ldots,\text{few}_{\text{student}}]}\lvert\text{student}\rvert$ $r_2=\frac{[1,\ldots,\text{all}_{\text{MIT student}}]}{[\textit{all}_{\text{paper to ACL}}]}\lvert\text{paper to ACL}\rvert\leq\frac{[1,\ldots,\text{few}_{\text{student}}]}{[1,\ldots,\text{all}_{\text{paper}}]}\lvert\text{paper}\rvert$	REVERSE
	S			
	D			
	I	✓		
	r_1	↑		
	r_2	↙		
	(m, m)		No student attended any seminar. $(\forall s)(\text{student}\ (s)\rightarrow(\exists r)(\text{seminar}\ (r)\rightarrow\neg\text{attended}\ (s,r)))$ $(\forall r)(\text{seminar}\ (r)\rightarrow(\forall s)(\text{student}\ (s)\rightarrow\neg\text{attended by}\ (r,s)))$ $r_1=\frac{[1]}{[\text{all}_{\text{student}}]}\lvert\text{student}\rvert\leq\frac{[1,\ldots,\text{all}_{\text{seminar}}]}{[1,\ldots,\text{all}_{\text{studentt}}]}\lvert\text{student}\rvert$ $r_2=\frac{[1,\ldots,\text{all}_{\text{student}}]}{[\textit{all}_{\text{seminar}}]}\lvert\text{seminar}\rvert\leq\frac{[1,\ldots,\text{all}_{\text{student}}]}{[1,\ldots,\text{all}_{\text{seminar}}]}\lvert\text{seminar}\rvert$	REVERSE
	S			
	D			
	I	✓		
	r_1	↑		
	r_2			
	(m, f)		No one read any newspaper. $(\forall p)(\text{person}\ (p)\rightarrow(\exists n)(\text{newspaper}\ (n)\rightarrow\neg\text{read}\ (p,n)))$ $(\forall n)(\text{newspaper}\ (n)\rightarrow(\forall p)(\text{person}\ (p)\rightarrow\neg\text{read by}\ (n,p)))$ $r_1=\frac{[1]}{[\text{all}_{\text{person}}]}\lvert\text{person}\rvert\leq\frac{[1,\ldots,\text{few}_{\text{newspaper}}]}{[1,\ldots,\text{all}_{\text{person}}]}\lvert\text{person}\rvert$ $r_2=\frac{[1,\ldots,\text{all}_{\text{person}}]}{[\textit{all}_{\text{newspaper}}]}\lvert\text{newspaper}\rvert\leq\frac{[1,\ldots,\text{all}_{\text{person}}]}{[1,\ldots,\text{few}_{\text{newspaper}}]}\lvert\text{newspaper}\rvert$	REVERSE
	S			
	D			
	I	✓		
	r_1	↑		
	r_2	↗		

	(f,1)		No faculty at U of W is chairing any conference. $(\forall f)(\text{faculty at U of W } (f) \rightarrow (\exists c)(\text{conference } (c) \rightarrow \neg \text{chairing } (f, c)))$ $(\forall c)(\text{conference } (c) \wedge (\forall f)(\text{faculty at U of W } (f) \rightarrow \neg \text{chaired by } (c, f)))$ $r_1 = \frac{[1]}{[\text{all}_{\text{faculty at U of W}}]} \lvert \text{faculty at U of W} \rvert \leq \frac{[1]}{[1, \ldots, \text{few}_{\text{faculty}}]} \lvert \text{faculty} \rvert$ $r_2 = \frac{[1, \ldots, \text{all}_{\text{faculty at U of W}}]}{[\text{all}_{\text{conference}}]} \lvert \text{conference} \rvert \leq \frac{[1, \ldots, \text{few}_{\text{faculty}}]}{[1]} \lvert \text{conference} \rvert$	REVERSE
	S			
	D			
	I	✓		
	r_1	↑		
	r_2	↗		
	(m, 1)		No chemist works for any bank. $(\forall c)(\text{chemist } (c) \rightarrow (\exists b)(\text{bank } (b) \rightarrow \neg \textit{worksfor } (c, b)))$ $(\forall b)(\text{bank } (b) \wedge (\forall c)(\text{chemist } (c) \rightarrow \neg \text{work for } (c, b)))$ $r_1 = \frac{[1]}{[\text{all}_{\text{chemist}}]} \lvert \text{chemist} \rvert \leq \frac{[1]}{[1, \ldots, \text{all}_{\text{chemist}}]} \lvert \text{chemist} \rvert$ $r_2 = \frac{[1, \ldots, \text{all}_{\text{chemist}}]}{[\textit{all}_{\text{bank}}]} \lvert \text{bank} \rvert \leq \frac{[1, \ldots, \text{all}_{\text{chemist}}]}{[1]} \lvert \text{bank} \rvert$	REVERSE
	S			
	D			
	I	✓		
	r_1	↑		
	r_2	↗		
	(1,1)		No student at U of W is giving any AI seminar at Dillon hall. $(\forall s)(\text{student at U of W } (s) \rightarrow (\exists r)(\text{AI seminar at Dillon hall } (r) \rightarrow \neg \textit{give } (s, r)))$ $(\forall r)(\text{AI seminar at Dillon hall } (r) \wedge (\forall s)(\text{student at U of W } (s) \rightarrow \neg \text{given by } (r, s)))$ $r_1 = \frac{[1]}{[\text{all}_{\text{student at U of W}}]} \lvert \text{student at U of W} \rvert \leq \frac{[1]}{[1]} \lvert \text{student} \rvert$ $r_2 = \frac{[1, \ldots, \text{all}_{\text{student at U of W}}]}{[\textit{all}_{\text{AI seminar at Dhillon hall}}]} \lvert \text{AI seminar at Dhillon hall} \rvert \leq \frac{[1]}{[1]} \lvert \text{seminar} \rvert$	REVERSE
	S			
	D			
	I	✓		
	r_1	↑		
	r_2	↗		

Table 3.10a: Results for sentences of the form $\text{No}[p_1, p_2, \ldots, p_n]C_1$ R any $[q_1, q_2, \ldots, q_n]C_2$ with examples.



No/Any	[1]	[Few]	[Many]
[1]	0	0	0
[Few]	0	0	0
[Many]	0	0	0

Table 3.10b: Results for the example sentences of the form $\text{No}[p_1, p_2, \ldots, p_n]C_1$ R any $[q_1, q_2, \ldots, q_n]C_2$ are depicted in the table.

The table below shows the notations used in the tables above, which show the results as per the discussion in sections 3.2 and 3.3.

Notation	Description
No/A	Sentence of type $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{A}\,[q_1, q_2, \ldots, q_n]C_2$
No/Every	Sentence of type $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{Every}\,[q_1, q_2, \ldots, q_n]C_2$
No/All	Sentence of type $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{All}\,[q_1, q_2, \ldots, q_n]C_2$
No/Any	Sentence of type $\text{No}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$
A/No	Sentence of type $\text{A}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{no}\,[q_1, q_2, \ldots, q_n]C_2$
Every/No	Sentence of type $\text{Every}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{no}\,[q_1, q_2, \ldots, q_n]C_2$
All/No	Sentence of type $\text{All}\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{no}\,[q_1, q_2, \ldots, q_n]C_2$
A/Any	Sentence of type $\text{A}\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$
A/All	Sentence of type $\text{A}\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{all}\,[q_1, q_2, \ldots, q_n]C_2$
Every/Any	Sentence of type $\text{Every}\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$
(1, m)	$QC(R, C_1, C_2) = \langle 1, \text{many} \rangle$
(m, f)	$QC(R, C_1, C_2) = \langle \text{many}, \text{few} \rangle$
(1, f)	$QC(R, C_1, C_2) = \langle 1, \text{few} \rangle$
(m, m)	$QC(R, C_1, C_2) = \langle \text{many}, \text{many} \rangle$
(f, 1)	$QC(R, C_1, C_2) = \langle \text{few}, 1 \rangle$
(m, f)	$QC(R, C_1, C_2) = \langle \text{many}, \text{few} \rangle$
(m, 1)	$QC(R, C_1, C_2) = \langle \text{many}, 1 \rangle$
(1, 1)	$QC(R, C_1, C_2) = \langle 1, 1 \rangle$
D	Direct reading of a sentence.
I	Indirect reading of a sentence.
r_1	Direct reading of numeric algorithm.
r_2	Indirect reading of numeric algorithm.
✓	Reading we select when a sentence is read in direct form.
↑	The most plausible scope reading selected by the numeric algorithm.
↗	Reading selected by the numeric algorithm having high degree of plausibility w.r.t most plausible reading.
↙	Reading selected by the numeric algorithm having low degree of plausibility w.r.t most plausible reading.
WITH	Go **WITH** the QC algorithm predictions.
REVERSE	**REVERSE** the QC algorithm predictions.
1	1 represents WITH. (Go **WITH** the QC algorithm predictions)
0	0 represents REVERSE. (**REVERSE** the QC algorithm predictions)

1e	Go WITH the QC algorithm (both are equally plausible, although they have entirely different meanings!)
[1] Or [few] Or [many]	Refers to m_1 or m_2 value in $QC(R, C_1, C_2) = \langle m_1, m_2 \rangle$

Table: 5 Description of notations used in all of the tables in section 3.4.

3.5 Extension to the QC algorithm

From the above results we see that certain type[1] of sentences (i.e., the notation $Q_1 [p_1, p_2, ..., p_n] C_1 \; R \; Q_2 [q_1, q_2, ..., q_n] C_2$ is used to represent a sentence) seem to produce the same result for the considered QC's. They can be distinguish in the following way:

For the following QC's

$QC(R, C_1, C_2) = \langle 1, many \rangle$
$QC(R, C_1, C_2) = \langle 1, few \rangle$
$QC(R, C_1, C_2) = \langle few, many \rangle$
$QC(R, C_1, C_2) = \langle few, few \rangle$
$QC(R, C_1, C_2) = \langle many, many \rangle$
$QC(R, C_1, C_2) = \langle many, few \rangle$
$QC(R, C_1, C_2) = \langle few, 1 \rangle$
$QC(R, C_1, C_2) = \langle many, 1 \rangle$

- The following type of sentences,

$$No \; [p_1, p_2, ..., p_n] C_1 \; R \; Q_2 [q_1, q_2, ..., q_n] C_2 \equiv \begin{bmatrix} No \; [p_1, p_2, ..., p_n] C_1 \; R \; every [q_1, q_2, ..., q_n] C_2 \\ No \; [p_1, p_2, ..., p_n] C_1 \; R \; all [q_1, q_2, ..., q_n] C_2 \end{bmatrix}$$

[1] For explanation purpose we represent a sentence with the notation $^{Q_1 \; C_1 \; R \; Q_2 \; C_2}$. Any variation on Q_1, Q_2 or R is considered a type of sentence. For example $^{No \; C_1 \; R \; Q_2 \; C_2}$ is a type of sentence, further $^{No \; C_1 \; R \; a \; C_2}$ is a type of sentence, further more $^{No \; C_1 \; \neg R \; a \; C_2}$ is another type of sentence.

produce the similar results (i.e., the same table) shown below.

No/Every, No/All	[1]	[Few]	[Many]
[1]	1e	1	1
[Few]	1e		1
[Many]	1e	1e	1

- **The following type of sentences**

$$Q_1\ [p_1, p_2, \ldots, p_n]C_1\ R\ \text{No}[q_1, q_2, \ldots, q_n]C_2 \equiv \begin{bmatrix}\text{every}\ [p_1, p_2, \ldots, p_n]C_1\ R\ \text{no}[q_1, q_2, \ldots, q_n]C_2 \\ \text{all}\ [p_1, p_2, \ldots, p_n]C_1\ R\ \text{no}[q_1, q_2, \ldots, q_n]C_2\end{bmatrix}$$

Produce similar results (i.e., the same table) shown below.

Every/No, All/No	[1]	[Few]	[Many]
[1]	1	1	1
[Few]	0		1
[Many]	0	0	1

- **The following type of sentences,**

$$a\ [p_1, p_2, \ldots, p_n]C_1\ \neg R\ Q_2\ [q_1, q_2, \ldots, q_n]C_2 \equiv [a\ [p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{all}\ [q_1, q_2, \ldots, q_n]C_2]$$

and

$$\text{No}\ [p_1, p_2, \ldots, p_n]C_1\ R\ Q_2\ [q_1, q_2, \ldots, q_n]C_2 \equiv [\text{No}\ [p_1, p_2, \ldots, p_n]C_1\ R\ \text{any}\ [q_1, q_2, \ldots, q_n]C_2]$$

Seem to produce similar results (i.e., the same table) shown below.

No/Any, A/All	[1]	[Few]	[Many]
[1]	0	0	0
[Few]	0	0	0
[Many]	0	0	0

3.5.1 General Rules

From the patterns recognized above we can formulate set of rules that determine the most plausible reading of a sentence with negation.

These general rules are,

For sentences $A[p_1, p_2,..., p_n]C_1 \neg R\ all[q_1, q_2,..., q_n]C_2$ and
$No[p_1, p_2,..., p_n]C_1\ R\ any[q_1, q_2,..., q_n]C_2$

No/Any, A/All	[1]	[Few]	[Many]
[1]	0	0	0
[Few]	0	0	0
[Many]	0	0	0

```
Push negation to the relation
(r1,r2) ← apply the scope QC algorithm
if ( ( Q1=='a' ∧ Q2=='all' ∧ negationOnVerb ) \/
     ( Q1=='no' ∧ Q2=='any' ∧ negationInNounPhrase)
   )
{
   if ( (QC=⟨1,many⟩) ∨ (QC=⟨1, few⟩) ∨ (QC=⟨few, many⟩ \/
        (QC=⟨many, few⟩) ∨ (QC=⟨few,1⟩) ∨ (QC=⟨many,1⟩) \/
        (QC=⟨many, many⟩) ∨ (QC=⟨1,1⟩)
      )

          r1 = 1 - r1;    // negate QC predictions
          r2 = 1 - r2;    // negate QC predictions
          // proceed as before...

}
```

For sentence $A\,[p_1, p_2, \ldots, p_n]C_1\ \neg R\ \text{any}\,[q_1, q_2, \ldots, q_n]C_2$

A/Any	[1]	[Few]	[Many]
[1]	0	0	1
[Few]	0		0
[Many]	0	0	0

Push negation to the relation

```
(r1,r2) ← apply the scope QC algorithm
if ( ( Q1=='a' ∧ Q2=='any' ∧ negationOnVerb ) )
{
   if ( (QC=⟨1, few⟩) ∨ (QC=⟨few, many⟩ \/
        (QC=⟨many, few⟩) ∨ (QC=⟨few,1⟩) ∨ (QC=⟨many,1⟩) \/
        (QC=⟨many, many⟩) ∨ (QC=⟨1,1⟩)
      )
         r1 = 1 - r1;    // negate QC predictions
         r2 = 1 - r2;    // negate QC predictions
         // proceed as before...

}
```

For sentence $A\,[p_1, p_2, \ldots, p_n]C_1\ R\ \text{no}\,[q_1, q_2, \ldots, q_n]C_2$

A/No	[1]	[Few]	[Many]
[1]	0	1	1
[Few]	0		0
[Many]	0	0	0

Push negation to the relation

```
(r1,r2) ← apply the scope QC algorithm
if ( ( Q1=='a' ∧ Q2=='no' ) ∧ negationInNounPhrase )
{
    if ( (QC=⟨many,many⟩) ∨ (QC=⟨few,many⟩) ∨ (QC=⟨1,1⟩)
         (QC=⟨many,few⟩) ∨ (QC=⟨few,1⟩) ∨ (QC=⟨many,1⟩))
       r1 = 1 - r1;    // negate QC predictions
       r2 = 1 - r2;    // negate QC predictions
       // proceed as before...
```

For sentences No $[p_1, p_2, ..., p_n]C_1$ R Every $[q_1, q_2, ..., q_n]C_2$ and
No $[p_1, p_2, ..., p_n]C_1$ R All $[q_1, q_2, ..., q_n]C_2$

No/Every, No/All	[1]	[Few]	[Many]
[1]	1e	1	1
[Few]	1e		1
[Many]	1e	1e	1

push negation to the relation
`(r1,r2)` ← **apply the scope QC algorithm**

```
if(Q1=='No'∧ (Q2=='every ∨ Q2=='all') ∧
negationInNounPhrase)
{
      // r1 = r1;  Go WITH the QC predictions
      // r2 = r2;  Go WITH the QC predictions
// proceed as before...
}
```

For sentence No $[p_1, p_2, ..., p_n]C_1$ R A $[q_1, q_2, ..., q_n]C_2$

No/A	$[m_2=1]$	[Few]	[Many]
$[m_1=1]$	1	1	1
[Few]	1		1
[Many]	1	1	1

push negation to the relation
`(r1,r2)` ← **apply the scope QC algorithm**

```
if(Q1=='No'∧ (Q2=='a') ∧
negationInNounPhrase)
{
      // r1 = r1;  Go WITH the QC predictions
      // r2 = r2;  Go WITH the QC predictions
// proceed as before...
}
```

For sentences Every $[p_1, p_2, ..., p_n]C_1$ R no $[q_1, q_2, ..., q_n]C_2$ and
All $[p_1, p_2, ..., p_n]C_1$ R no $[q_1, q_2, ..., q_n]C_2$

Every/No, All/No	[1]	[Few]	[Many]
[1]	1	1	1
[Few]	0		1
[Many]	0	0	1

Push negation to the relation

```
(r1,r2) ← apply the scope QC algorithm
if ( ( Q1=='every' ∨ Q1=='all' ) ∧ Q2=='no' ) )
      ∧ negationInNounPhrase )
{
    if ((QC=⟨many,few⟩) ∨ (QC=⟨few,1⟩) ∨ (QC=⟨many,1⟩))
       r1 = 1 - r1;    // negate QC predictions
       r2 = 1 - r2;    // negate QC predictions
       // proceed as before...
```

For a sentence Every $[p_1, p_2, ..., p_n]C_1$ $\neg$R any $[q_1, q_2, ..., q_n]C_2$

Every/Any	[1]	[Few]	[Many]
[1]	1	1	1
[Few]	0		1
[Many]	0	0	1

Push negation to the relation

```
(r1,r2) ← apply the scope QC algorithm
if ( Q1=='every' ∧ Q2=='any' ∧ negationOnVerb )
{
    if ((QC=⟨many,few⟩) ∨ (QC=⟨few,1⟩) ∨ (QC=⟨many,1⟩))
       r1 = 1 - r1;    // negate QC predictions
       r2 = 1 - r2;    // negate QC predictions
       // proceed as before...
}
```

Chapter 4

Natural Language Interpreter For Quantifier Scope Ambiguity Resolution

4.1 Introduction

The purpose of this study is to investigate an extension to Saba, (1999) QC algorithm in such a way so as to allow the use of negation in a sentence. The claim that we are defining is that the extension to the QC algorithm of Saba, (1999) as described in the previous chapter extends the capability of QC algorithm by allowing the use of negation in a sentence.

To investigate the extension to the QC algorithm of Saba, (1999) we have incorporated changes in the existing natural language interpreter for resolution of quantifier scope ambiguity so as to account for negation in a sentence. The interpreter is capable of resolving the quantifier scope ambiguity. This interpreter parses a sentence with quantifiers to give scope neutral logical form and determines the most plausible scope reading. This interpreter is implemented in JAVA.

4.2 Interpreter for Quantifier Scope Ambiguity Resolution

The interpreter for Quantifier scope ambiguity resolution was implemented by Saba, 1999. For a sentence

(4.1) $(S(NP(DET\ Q_1)(TP[p_1,...,p_m]X))(VP(v\ R)(NP(DET\ Q_2)(TP[q_1,...,q_n]Y))))$

The interpreter is aimed at generating the following:

- The parse structure (i.e., the syntactic structure).

- The scope neutral logical form.
- Determines the most plausible scope ordering.

That is for the sentence below the interpreter generates the following.

(4.2) Every student attended a seminar

(4.2.1) Parse structure:
[S [NP [DET Every][TP student]]
[VP [VERBtr attended][NP [DET a][TP seminar]]]]

(4.2.2) Scope Nuetral Logical Form:

$$\left[\begin{array}{l}\text{FORALL(x1)}\left[\begin{array}{l}\text{STUDENT(x1)}\rightarrow\\ \text{EXISTS(x2)[SEMINAR(x2)\& Attended(x1, x2)]}\end{array}\right]\\ \text{EXISTS(x1)}\left[\begin{array}{l}\text{SEMINAR(x1)\&}\\ \text{FORALL(x2)[STUDENT(x2)}\rightarrow\text{ Attended(x1, x2)]}\end{array}\right]\end{array}\right]$$

(4.2.3) Most Plausible Reading:

Retrieved the relevant $QC(1^+, 1^+)$
PLAUSIBILITY for direct reading = 1.0
Retrieved the relevant $QC(1^+, 1^+)$
PLAUSIBILITY for indirect reading = 0.01
Direct reading is most plausible
Most plausible reading is

$$\text{FORALL(x1)}\left[\begin{array}{l}\text{STUDENT(x1)}\rightarrow\\ \text{EXISTS(x2)[SEMINAR(x2)\& Attended(x1, x2)]}\end{array}\right]$$

In the following sub sections we discuss implementation details of the interpreter in detail.

4.2.1 Parsing a Sentence

A sentence is made up of linguistic objects (i.e., syntactic objects). The syntactic structure of a sentence involves non-terminal and terminal linguistic objects. These linguistic objects are implemented as classes shown below. In this section we discuss the implementation details of the parser for this interpreter. Each linguistic object (i.e.,

syntactic object) has a meaning object (i.e., semantic object). Thus the parser obtains the meaning of each linguistic object for a successfully parsed sentence.

• Non - terminal Linguistic Objects :	Classes
Sentence	_SENTENCE
Noun Phrase	_NP , _NP0
Verb Phrase	_VP
Transitive Pronoun	_TP
Noun	_NOUNS
Adjective	_ADJS
Prepositional Phrase attachment	_PP
• Terminal Linguistic Objects :	Classes
Determiner	_DET
Common Noun	_NOUN
Proper Noun	_PNOUN
Adjective	_ADJ
Preposition	_PREP
Transitive Verb	_TVERB

Each of these classes is a subclass of Class _LingObj.

The following are some rules that are applicable to the non-terminal Linguistic classes. That is the syntactic structure of a non-terminal linguistic object.

For Class _SENTENCE
Rule (np, vp)
a sentence (obj) is make up of two linguistic objects a noun phrasr (np) followed by a verb phrase (vp).

For Class _NP
Rule (np0a, pp)
a noun phrase (np) is make up of two linguistic objects a noun phrase (np0a) followed by a prepositional phrase (pp).
Rule (np0c)
a noun phrse (np) is simply a noun phrase (np0c) (i.e., involves a determinar followed by a term phrase)
For Class _NP0
Rule (det, tp)
a noun phrase (np0) is make up of two linguistic objects a determinar (det) followed by a term phrase (tp)
Rule (pnoun)
a noun phrse (np0) is a proper noun (pnoun)

For Class _TP
Rule (Adjs, nouns)
a term phrase (tp) is made up of two linguistic objects Adjectives (Adjs) followed by common nouns (nouns)
Rule (nouns)
a term phrase (tp) is a common noun (nouns).
Rule (pnoun)
a tterm phrase (tp) is a proper noun (pnoun).

For Class _VP
Rule(tverb, np)
a verb phrase (vp) is made up of two linguistic objects a transitive verb (tverb) followed by a noun phrase (np)

For Class _PP
Rule (prep, np)
a prepositional phrase (pp) is made up of two linguistic objects a preposition (prep) followed by a noun phrase (np)

A non-terminal linguistic object can have more than one syntactic structure. The various syntactic structures for the same linguistic object are handled by the method public `Vector orElse( Vector words, Vector cases ..... )` in `class LingObj`. The various syntactic structures for the same linguistic object are the rules of the respective class discussed above. The code below shows how these rules are handled for a non-terminal linguistic object (i.e., Noun Phrase (NP), Term Phrase (TP)...).

For Class _NP

$$\text{orElse}\begin{pmatrix}\text{input, rule(np0a, pp),}\\ \text{rule(np0b, rclause),}\\ \text{rule(np0c)}\end{pmatrix}$$

that is a noun phrase (np) can have a structural break down as
noun phrase (np0a) followed by a prepositional phrase
or noun phrase (np0b) followed by a relative clause
or simply a noun phrase (i.e., a transitive pronoun)

For Class _NP0

orElase(input, rule(det, tp),
rule(pnoun))

that is a noun phrase (np) can have a structural break down as
a determinar (det) followed by a term phrase (tp)
or a proper noun (pnoun)

For Class _TP

orElse(input, rule(adjs, nouns),
rule(nouns),
rule(pnoun))

that is a term phrase can have a structural break down as
adjectives (adjs) followed by common nouns (nouns)
or common nouns (nouns)
or a proper noun (pnoun)

The main point here is to show that each non-terminal linguistic object can have as its structure any one of the various rules (i.e., syntactic structure of a linguistic object) applicable to that linguistic object.

These various applicable rules (i.e., syntactic structure of a linguistic object) for non-terminal linguistic object make up the complete syntactic structure for a sentence as in (4.1). that is these non terminal linguistic objects ane made up of linguistic objects (i.e., syntactic objects). Here we avoid discussing the relative clause.

4.2.1.1 Attributes Defined

The following are the global attributes that are defined.

```
Vector    input
Vector    restOfInput
boolean   parsedOk
Vector    tokens
String    parseTree
int       numOfNodes = 0
```

```
a class (global) attributes pointing to the dictionary
final static Dictionary  dictionary = new Dictionary()
every linguistic object has a meaning
Meaning  meaning  =  new Meaning()
```

These attributes keep track of the input sentence to be parsed, the remaining input sentence to be parsed, the successfulness of the parse, collected tokens from the successful parses, number of terminal nodes parsed, the syntactic structure of the parse tree and the meaning of each linguistic object at each stage (i.e., syntactic level) of the parse. Thus these attributes determine the success or failure of a sentence parse. In case of a successful parse the syntactic structure of the sentence is obtained and the meaning of each linguistic object is initialized. In later stages of implementation where we obtain the meanings of a sentence (i.e., scope neutral logical form) and to determine the most plausible scope reading these meaning objects are referred.

The important note point here is that each *non-terminal linguistic object* class has attributes defined for linguistic objects (i.e., the linguistic objects of the syntactic structure at that specific linguistic object) and meaning object (i.e., the meaning of that specific linguistic object). And each terminal linguistic object has only a meaning attribute. Following compositional semantics, from the syntactic structure of an English sentence we see that the meaning for each linguistic object is obtained by assigning meaning or by forming an expression (i.e., meaning of noun phrase 'every student' is an expression $\text{FORALL}(X1)(\text{STUDENT}(X1) \rightarrow Q1(X1))$ and meaning of student is STUDENT (meaning assigned from the dictionary for the meaning of a noun)).

In appendix B we give example code for terminal and non-terminal linguistic objects (i.e., syntactic objects) of an English sentence implemented as a class. The example code for the linguistic object shows the attributes defined, applicable syntactic structure of the linguistic object, process of determining the meaning for the linguistic

object and process of determining the success or failure of a parse at that node (i.e., syntactic level) by assigning values to the attributes that are global and defined in the class.

4.2.1.2 Meaning of Linguistic Objects in a Sentence

Linguistic Objects (i.e., syntactic objects) of a sentence are implemented as classes. These classes are mentioned in section 4.2.1. The meaning of a Linguistic Object is assigned using the classes shown below. Brief discussion of meanings for linguistic objects of a sentence with their implementation details is given below.

Linguistic Objects	Classes
(1) Quantifiers (Numaric or Linguistic Quantifiers)	Quantifier
(2) Noun Modifiers (Adjective)	Modifier
(3) Concept (Noun)	Concept
(4) noun Attachment (Preposition followed by a noun phrase (i.e., prepositional phrase attachment))	Attachment
(5) Quantified Concept (Noun Phrase)	QuantifiedConcept
(6) Clause of a Quantified Concept (Verb Phrase is in the clause of a quantified concept (i.e., noun phrase))	Clause
(7) Verb (Relation)	Relation
(8) Sentence (*Sentence*)	LogicalForm

Each of these classes has `Class Meaning` as the super class. These classes assign meanings to the linguistic objects of a sentence either by referring to the Class Dictionary or forming an expression using `class Expression1`. That is *usually* the meaning of terminal linguistic objects, like proper noun, is obtained by referring to the `Class`

`Dictionary` (i.e., `Concept meaning = dictionary.getMeaningOfPNoun (word)`) and the meaning of a non-terminal linguistic object, like quantified concept (i.e., a noun phrase), is assigned by forming an expression using `Class Expression1`.

Here the terminal linguistic object preposition and determiner are assigned meanings in their respective classes (i.e., `Class _PREP and Class _DET` respectively). Some non-terminal linguistic objects like Nouns, Adjectives, Prepositional Phrases, etc., each have an attribute of type vector for collecting an occurrence of them whose meaning is assigned at their respective terminal linguistic object (the syntactic structure of these linguistic objects has a terminal linguistic object). We also note that in forming an expression the quantified concept uses the noun modifiers (i.e., Nouns and Adjectives) and noun attachments (i.e., prepositional phrase) as conjunction to the concept (i.e., noun).

4.2.1.3 Example Sentence

From the structure of an English sentence it is noticeable that each syntactic object has a semantic object. The parser of this interpreter is implemented by this method that is each linguistic object has a meaning object.

The successful parse of a sentence gives

- The parse structure (i.e., syntactic structure).

And initializes

- Meaning object of each linguistic object.

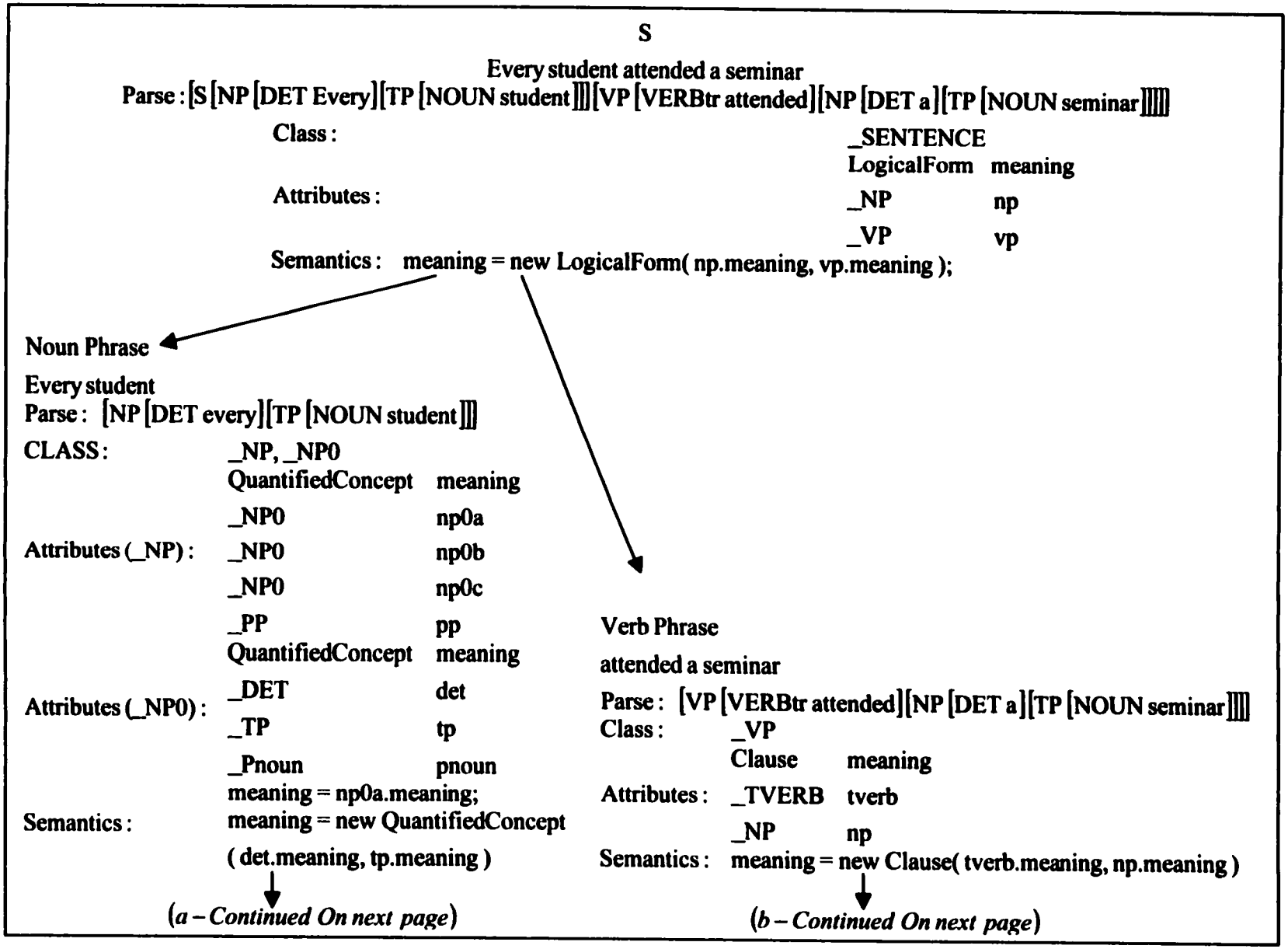
For example consider the following sentence:

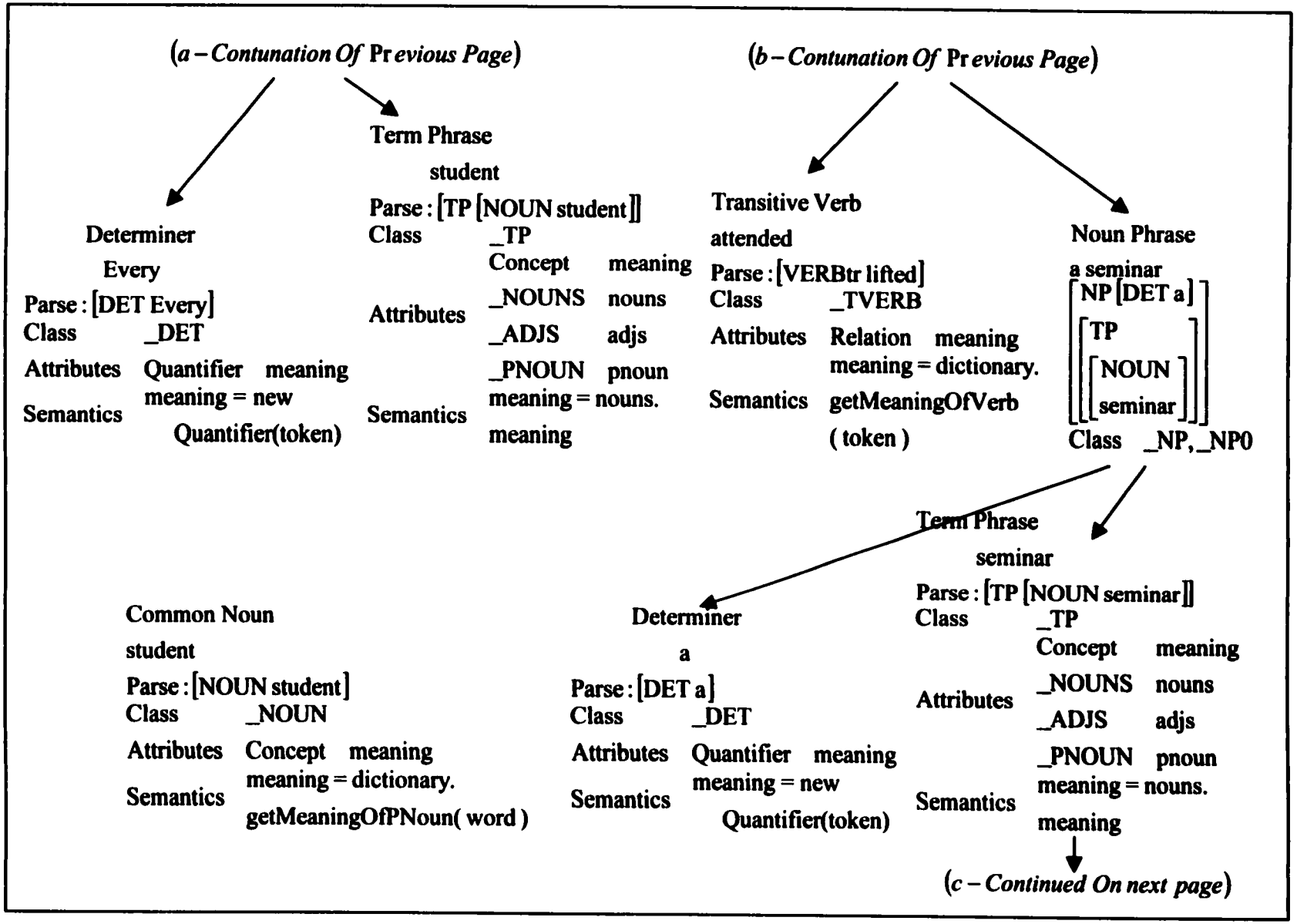
- Every student attended a seminar.

The figure (3) is the syntactic structure of the sentence (4.1) given by the parser of this interpreter and illustrates the following about a sentence.

- The parse structure (i.e., the syntactic structure) of an English sentence.
- Linguistic objects (i.e., syntactic objects) are implemented as classes.
- Each linguistic object has a meaning object.
- The meaning object of their respective meaning class initializes the meaning of each linguistic object.
- The attributes defined for terminal and non-terminal linguistic objects.

The details about the notations used in figure (3) are given in table (6).


S
Every student attended a seminar
Parse : [S [NP [DET Every][TP [NOUN student]]][VP [VERBtr attended][NP [DET a][TP [NOUN seminar]]]]
Class : _SENTENCE
Attributes : LogicalForm meaning
_NP np
_VP vp
Semantics : meaning = new LogicalForm(np.meaning, vp.meaning);
Noun Phrase
Every student
Parse : [NP [DET every][TP [NOUN student]]]
CLASS : _NP, _NP0
Attributes (_NP) : QuantifiedConcept meaning
_NP0 np0a
_NP0 np0b
_NP0 np0c
_PP pp
Attributes (_NP0) : QuantifiedConcept meaning
_DET det
_TP tp
_Pnoun pnoun
Semantics : meaning = np0a.meaning;
meaning = new QuantifiedConcept
(det.meaning, tp.meaning)
(a – Continued On next page)
Verb Phrase
attended a seminar
Parse : [VP [VERBtr attended][NP [DET a][TP [NOUN seminar]]]
Class : _VP
Attributes : Clause meaning
_TVERB tverb
_NP np
Semantics : meaning = new Clause(tverb.meaning, np.meaning)
(b – Continued On next page)


(a – Contunation Of Previous Page)
(b – Contunation Of Previous Page)
Term Phrase
student
Parse : [TP [NOUN student]]
Class _TP
Attributes Concept meaning
_NOUNS nouns
_ADJS adjs
_PNOUN pnoun
Semantics meaning = nouns.
meaning
Determiner
Every
Parse : [DET Every]
Class _DET
Attributes Quantifier meaning
Semantics meaning = new Quantifier(token)
Transitive Verb
attended
Parse : [VERBtr lifted]
Class _TVERB
Attributes Relation meaning
Semantics meaning = dictionary. getMeaningOfVerb (token)
Noun Phrase
a seminar
[NP [DET a] [TP [NOUN seminar]]]
Class _NP, _NP0
Term Phrase
seminar
Parse : [TP [NOUN seminar]]
Class _TP
Attributes Concept meaning
_NOUNS nouns
_ADJS adjs
_PNOUN pnoun
Semantics meaning = nouns.
meaning
Determiner
a
Parse : [DET a]
Class _DET
Attributes Quantifier meaning
Semantics meaning = new Quantifier(token)
Common Noun
student
Parse : [NOUN student]
Class _NOUN
Attributes Concept meaning
Semantics meaning = dictionary. getMeaningOfPNoun(word)
(c – Continued On next page)

(*c – Continuation Of* Pr*evious Page*)

Common Noun

seminar

Parse : [NOUN seminar]

Class _NOUN

Attributes Concept meaning

Semantics meaning = dictionary.
getMeaningOfNoun(word)

Figure: 3 Parse structure of an example sentence by the interpreter for Quantifier Scope Ambiguity Resolution.

Notations Used For Parsing a sentence in Figure:1	Table: 6
Class	• A sentence is made up of linguistic objects. Each linguistic object is implemented as a class (i.e., Noun phrase ≡ _NP, Proper noun ≡ _Pnoun).
Attributes	• Non-terminal linguistic object classes have attributes for linguistic objects that form syntactic structure at that syntactic level and meaning attribute that is an object of *its* respective meaning class. • Terminal linguistic object classes have attributes only for meaning that is an object of *its* respective meaning class.
Semantics	• Meaning (Semantics) of a non-terminal linguistic object is stored in its respective meaning class. Also for this linguistic object an expression may be made where required by referring to the terminal and non-terminal linguistic objects, which form its syntactic structure. (i.e., `meaning of noun phrase : meaning = new QuantifiedConcept( det.meaning, tp.meaning ; meaning.makeExpression( 0 ))` • Meaning (Semantics) of a terminal linguistic object is *usually* referred to the dictionary and stored in its respective meaning class. (i.e., meaning of proper noun: `meaning = dictionary.getMeaningOfPNoun( word )`)
Parse	• Gives the syntactic structure of a linguistic object.

4.2.2 Generating Scope Neutral Logical Form

A sentence is a linguistic object (i.e., a syntactic object). The meaning of a sentence is its logical form. For example the meaning of a sentence in (4.2) repeated below is

Every student attended a seminar.

(4.3) FORALL(X1)[STUDENT(X1)→EXISTS(X2)[SEMINAR(X2)& Attended(X1, X2)]]
(4.4) EXISTS(X1)[SEMINAR(X1)& FORALL(X2)[STUDENT(X2)→ Attended(X1, X2)]]

The above sentence involves two quantifiers thus this sentence has two meanings one with the wide scope 'every' and the other with the wide scope 'a'. Brief discussion of the implementation details for obtaining the meanings of a sentence as in (4.1) is given below.

The linguistic object sentence is implemented as `Class _SENTENCE`. Each linguistic object has a meaning object. Thus a sentence has a meaning object (i.e., semantic object), which is implemented as `Class LogicalForm`. Below is the code of this class showing the attributes defined and some methods for generating the meanings for a sentence.

During the parse at the syntactic level[5] of the sentence, its meaning object calls the constructor of the meaning class thus initializing meaning object to the following attributes:

Meaning object of a sentence : meaning = new LogicalForm (np.meaning, vp.meaning);

```
Relation            relation;     // Initialized to tverb.meaning[6]
QuantifiedConcept   qConcept1;    // Initialized to np.meaning[7]
QuantifiedConcept   qConcept2;    // Initialized to np.meaning[8]
```

[5] Execution of the method `public void parse ()` in a subclass of `class _LINGOBJ`.
[6] For a sentence [S[NP_1][VP[$_vR$][NP_2]] attribute `relation` is initialized to meaning of $_vR$
[7] For a sentence [S[NP_1][VP[$_vR$][NP_2]] attribute `qConcept1` is initialized to meaning of NP_1

Basically the structure of a sentence as in (4.1) is

(4.5) $(S(NP)(VP(R)(NP)))$

A noun phrase may involve a determiner followed by term phrase. This sentence involves quantifier scope ambiguity. By the method of quantifier rising or quantifying in two structures for the same sentence are obtained.

(4.6) $(S(NP_1)(VP(R)(NP_2)))$
(4.7) $(S(NP_2)(VP(R)(NP_1)))$

That is this sentence has two meanings. The attributes `qConcept1` and `qConcept2` are initialized to meaning objects of noun phrases and `relation` is initialized to meaning objects of the verb.

```
class LogicalForm extends Meaning
{
        Relation                    relation;
        QuantifiedConcept           qConcept1;
        QuantifiedConcept           qConcept2;


        LogicalForm( QuantifiedConcept qc, Clause c )
        {
                relation = c.relation;
                qConcept1 = qc;
                qConcept2 = c.qConcept;
        }

        public String getLogicalForm( QuantifiedConcept qConcept1,
QuantifiedConcept qConcept2 )
        {
                Expression1 expr1a = qConcept1.getExpression(0);
                Expression1 expr2a = qConcept2.getExpression(1);
                expr2a.replaceBinaryRelation( relation.relation );
                expr1a.applyExpression( expr2a );
                return expr1a.getMatrix();
        }

        public String getScopeNeutralLF()
        {
                String finalRes = new String();
                finalRes = finalRes + "[\n";
                finalRes = finalRes + "\t"
                             + getLogicalForm(qConcept1,qConcept2) + ";\n\t"
                             + getLogicalForm(qConcept2,qConcept1);
                finalRes = finalRes + "\n]\n";
                return finalRes;
        }
}
```

[8] For a sentence $[S[NP_1][VP[_vR][NP_2]]$ attribute `qConcept2` is initialized to meaning of NP_2

The method `getScopeNeutralLF()` calls `getLogicalForm ( QuantifiedConcept qConcept1, QuantifiedConcept qConcept2 )` twice each time swapping the noun phrases thus obtaining the meanings for a sentence. The method `getLogicalForm (QuantifiedConcept qConcept1, QuantifiedConcept qConcept2)` obtains the meaning of a sentence (4.1). Here we see that meaning of a sentence depends on the position of the noun phrase as in (4.6) and (4.7).

Methods used for forming the meaning for a noun phrase and meanings of a sentence are given below.

```
public Expression1 getExpression(int varCount)          of Class QuantifiedConcept
public void makeExpression( int varCount )              of Class QuantifiedConcept
Expression1(int vCount, Quantifier quant, Concept concept ) of Class Expression
public void applyExpression( Expression1 expr )         of Class Expression
public void replaceBinaryRelation( String rel )         of Class Expression
public String getMatrix()                               of Class Expression
```

In order to obtain the meaning of a sentence[9], first the meaning of each noun phrase is obtained and then the meaning of one noun phrase is combined[10] with the meaning of a verb to form an expression, this expression is the meaning of verb phrase, Finally the meaning of other noun phrase is combined with the meaning of verb phrase. The methods used in obtaining meanings of a sentence are explained in table 7 and code for some of these methods is given. In chapter 5 we discuss these methods with an example sentence.

[9] For explanation purpose we consider sentence $[S[NP_1][VP[_VR][NP_2]]$ where each noun phrase (i.e., NP1 and NP2) has a determiner.

[10] For a sentence $[S[NP_1][VP[_VR][NP_2]]$ meaning of VP is obtained form the meaning of $_VR$ and NP_2

Class LogicalForm extends Meaning

```
Relation                relation;
QuantifiedConcept       qConcept1;
QuantifiedConcept       qConcept2;
```

`LogicalForm( QuantifiedConcept qc, Clause c )`

The constructor initializes the above attributes during the parse at the syntactic level of the sentence. These attributes are the meaning (i.e., semantic) objects of the linguistic (i.e., syntactic) objects, noun phrases and verb, which make up a sentence.

`Public String getScopeNeutralLF()`

This method obtains the meanings of a sentence as in (4.1). Meaning of a sentence is an expression made up of two noun phrases and a verb. Since a sentence such as (4.1) involves quantifiers there could be more than one meaning for the same sentence, which vary in the position of the noun phrase. A call is made to the method `public String getLogicalForm( QuantifiedConcept qConcept1, QuantifiedConcept qConcept2 )` that obtains meaning of the sentence (i.e., by varying the positions of the noun phrases).

`Public String getLogicalForm( QuantifiedConcept qConcept1, QuantifiedConcept qConcept2 )`

This method obtains the meaning of a scope reading for a sentence as in (4.1) which depends on the passed parameters qConcept1 and qConcept2 (i.e., the position of the noun phrases in a sentence). Basically in this method meaning for each noun phrase is obtained and then the meaning of one noun phrase is combined with the relation thus forming the meaning of verb phrase and finally the meaning of other noun phrase is combined with the meaning of the verb phrase to obtain the meaning of a sentence. We briefly state the method calls below.
The method `getExpression(0 or 1)` forms an expression (i.e., meaning) for each noun phrase.
The method `replaceBinaryRelation( relation.relation )` replaces the binary relation for a predicate in the meaning of a noun phrase thus obtaining meaning for verb phrase.
The method `applyExpression (expr2a)` unifies the meaning of a noun phrase with the meaning of a verb phrase thus forming a single expression (i.e., meaning), which is the meaning of a sentence.
The method `getMatrix()` retrieves the meaning of a sentence for a scope reading.

Class QuantifiedConcept extends Meaning

```
Concept                 concept;
Quantifier              quantifier;
Attachment              attachment;
Expression1             expression1;
```

```
QuantifiedConcept( Quantifier quant, Concept con )
```

The constructor initializes the above attributes (i.e., quantifier and concept) during the parse at the syntactic level of the noun phrase. These attributes are the meaning (i.e., semantic) objects of the linguistic (i.e., syntactic) objects determiner and noun respectively. A determiner (optional), adjectives (optional), noun(s), prepositional phrase (optional) form a noun phrase.

```
Public Expression1 getExpression(int varCount)
```

This method obtains the meaning of a noun phrase involving a quantifier and a concept. The meaning of a noun phrase is an expression.

```
Public void makeExpression( int varCount )
```

This method adds up any concept modifiers (adjectives, nouns, prepositions, etc.,) for forming the meaning of a noun phrase.

Below we give the code for the method getExpression( )

```
Public Expression1 getExpression(int varCount)
{                       Expression1 expr = new Expression1(varCount,quantifier,concept);
             makeExpression(varCount);
             expr.quantifier = expression1.quantifier;
             expr.variable = expression1.variable;
             expr.predicate1 = expression1.predicate1;
             expr.predicate2 = expression1.predicate2;
             expr.connective = expression1.connective;
             return expr;
}
```

Class Expression1 extends Expression

```
int          varCount = 0;
int          nextPredicate = 1;

String quantifier = new String ("");
String variable = new String ("");
String predicate1 = new String ("");
String predicate2 = new String ("");
String connective = new String ("");

String matrix = new String ("");
```

`Expression1( int vCount, Quantifier quant, Concept concept )`

The constructor forms a meaning (i.e., expression) for a noun phrase or a sentence. This meaning is stored in the attributes defined above.

`Public void replaceBinaryRelation( String rel )`

This method replaces the attribute `predicate2` with meaning of verb (relation between two concepts) in the meaning of a noun phrase thus obtaining the meaning of verb phrase.

`public void applyExpression( Expression1 expr)`

This method combines the meaning of a noun phrase with the meaning of a verb phrase thus forming the meaning of a sentence.

`Public String getMatrix()`

This method returns meaning for a noun phrase or a sentence.

Below we give the code for the method `replaceBinaryRelation ()`

```
Public void replaceBinaryRelation (String rel)
      {
            predicate2 = new String();
```

```
            predicate2 = rel + variable;


            String oldPred2 = new String("");
            oldPred2 = predicate2;
            int len = oldPred2.length();
            oldPred2 = oldPred2.substring(0,len-3)
                        + expr.variable.substring(1,3) + ","
                        + oldPred2.substring(len-3);

            predicate2 = oldPred2;
      }
```

Below we give the code for the method `applyExpression( Expression1 expr)`

```
public void applyExpression( Expression1 expr)
{
                if ( nextPredicate == 1 ) predicate1 = expr.getMatrix();
                     else predicate2 = expr.getMatrix();
}
```

Table: 7 **Explanation for class methods that obtain meanings of a sentence.**

These methods are explained with an example in the chapter 5.

4.2.3 Determining the Most Plausible Scope Reading

For a sentence as in (4.1) plausibility of each scope reading has to be determined as two readings may not be equally plausible. A numerical algorithm that determines the degree of plausibility of each scope reading is given by Saba, 1999. Determining the most plausibility scope reading for a sentence as in (4.1) is implemented as a method `scope ()` of `Class LogicalForm`, at the meaning level[11] of a sentence. The code for this method is given in appendix B.

We repeat the numerical algorithm below.

$$(4.8)\qquad \begin{aligned} r_1 &= 1 \le \frac{\text{Range}(Q_2, C_2)}{\text{Range}(Q_1, C_1)} |X| \le \frac{[1,\ldots,m_2]}{[1,\ldots,m_1]} |C_1| \\ r_2 &= 1 \le \frac{\text{Range}(Q_1, C_1)}{\text{Range}(Q_2, C_2)} |Y| \le \frac{[1,\ldots,m_1]}{[1,\ldots,m_2]} |C_2| \end{aligned}$$

The method `scope ()` obtains degree of plausibility of the direct and indirect readings by passing the attributes `qConcept1`, `qConcept2` (i.e., meaning objects of noun phrases) and the order (i.e., indication for direct or indirect reading) to the method `scopeTop(QuantifiedConcept quantCon1, QuantifiedConcept quantCon2, int order)`. The obtained plausibility measures for the direct and indirect readings are compared to determine the most plausible scope reading.

The formula below discussed in detail in chapter-2 is implemented as the method `public double computePlausibilityMeasure(Vector range1, Vector range2 )`. This method determines the degree of plausibility of a scope reading.

$$(4.9)\qquad p(I,J) = \Pr(I,J)|J| + \sum_{i=1}^{|I\cap J|} \left(|J| - \Pr(J,I) - i + 1\right) |I \cap J| \left(\frac{1}{|I|\ |J|}\right)$$

Where,

$$\Pr(I,J) = |\{i \in I \mid i < J_{min}\}|$$

[11] For a sentence each syntactic object has a semantic object. The syntactic object sentence is implemented as `Class _SENTENCE` and its meaning object is implemented as `Class LogicalForm`.

Some important methods that determine the most plausible reading for a sentence (4.1) are explained in table 8. The table explains only the functionality of important methods[12] in the `class LogicalForm` in terms of obtaining the ranges I and J in (4.9) from the numerical algorithm (4.8) and computes the plausibility measure p (I, J) (see Appendix B for code). I avoid mentioning and explaining each method[13] that is implemented to construct ranges I and J and measure the degree of plausible of a scope reading.

This chapter explained the implementation details of the Natural Language interpreter for Quantifier Scope Ambiguity Resolution. In the next chapter we explain the process of incorporating negation into this system.

[12] The methods that determine the most plausible reading, obtain ranges for I and J, compute plausibility measure.
[13] The methods used (i.e., called) by ones mentioned in 8.

```
Class LogicalForm extends Meaning

      Relation                    relation;
      QuantifiedConcept           qConcept1;
      QuantifiedConcept           qConcept2;
      Vector                      readings = new Vector ();
```

```
Public void scope ()

      Vector reading1 = new Vector ();
      Vector reading2 = new Vector ();
      double r1;
      double r2;
```

This method obtains a measure of plausibility for the direct and the indirect readings (r1 and r2) of a sentence (4.1) by passing on as parameters the meaning objects of noun phrases (i.e., qConcept1 and qConcept2) and an integer value (1 and 0 indicate direct reading and indirect reading respectively) to the method `public double scopeTop( QuantifiedConcept quantCon1, QuantifiedConcept quantCon2, int order )`. For each readings plausibility measure the meaning objects of noun phrases are swapped and passed as parameters. The measure of plausibility in r1 and r2 are compared to determine the most plausible scope reading.

```
Public double scopeTop( QuantifiedConcept quantCon1, QuantifiedConcept quantCon2, int order )

            Concept           concept1 = quantCon1.concept;
            Concept           concept2 = quantCon2.concept;
            Quantifier        quantifier1 = quantCon1.quantifier;
            Quantifier        quantifier2 = quantCon2.quantifier;

            double      conceptX = quantCon1.getModifiersEffect();
            double      conceptY = quantCon2.getModifiersEffect();
            double      ratioA = conceptX/quantCon1.concept.typicalCardinality;
            double      ratioB = conceptY/quantCon2.concept.typicalCardinality;

            Vector      rangeV1 = constructRange (qc.getFirst (), quantCon1.concept);
            Vector      rangeV2 = constructRange (qc.getSecond (), quantCon2.concept);
            Vector      rangeQ1 = constructRange (quantifier1.quantifier, quantCon1.concept);
            Vector      rangeQ2 = constructRange (quantifier2.quantifier, quantCon2.concept);

            Vector      finalRange1 = divideRanges(rangeQ2,multiplyRange(ratioA,rangeQ1));
```

`Vector finalRange2 = divideRanges(rangeV2,rangeV1);` This method obtains the ranges I and J that are used to determine the measure of plausibility for a scope reading by the formula p (I, J) in (4.9) which is implemented as the method `public double computePlausibilityMeasure( Vector range1, Vector range2 )`. The ranges I and J are obtained as shown in (4.8), that is I = [Range (Q_2, C_2)/Range (Q_1, C_1)) * ($|X|/|C_1|$] and J = [[1,...,m_2]/[1,...,m_1]] for direct reading and vice versa for in direct reading. Basically the attributes defined above are initialized to a value or range in (4.8) which is sketched below. `double ratioA` = $|X|/|C_1|$ `double ratioB` = $|Y|/|C2|$ `Vector rangeV1` = [1,...,m_2] if order ==1 (i.e, for direct reading) = [1,...,m_1] if order == 0 (i.e, for indirect reading) `Vector rangeV2` = [1,...,m_1] if order ==1 (i.e, for direct reading) = [1,...,m_2] if order ==0 (i.e, for indirect reading) `Vector rangeQ1` = Range (Q_i, C_i) `Vector rangeQ2` = Range (Q_i, C_i) `Vector finalRange1` = [Range (Q_i, C_i)/Range (Q_i, C_1)) * ($|X_i|/|C_i|$] } where $1 \le i \le 2$. Note for each scope reading the parameters which are meaning objects of noun phrases are swapped. `Vector finalRange2` = [[1,...,m_2]/[1,...,m_1]] The vectors `finalRange1` and `finalRange2` that are range of values I and J respectively are passed as parameters to the method `public double computePlausibilityMeasure (Vector range1, Vector range2).`
`public double computePlausibilityMeasure( Vector range1, Vector range2 )` This method calculates the measure of plausibility p (I, J) in (4.9).

Table 8: Explanation for class methods that obtain Most Plausible Scope Reading of a sentence.

CHAPTER 5

Details Of Implementation

In the last chapter the general framework for the Natural Language Understanding interpreter for Quantifier scope ambiguity resolution was briefly described. This chapter discuss as how this interpreter handles negation and computes the meaning of a sentence with some examples.

5.1 Presence Of Negation in a Sentence

A sentence as in (4.1), repeated below, may have a negation word as the determiner in a noun phrase or negation on the verb. We are interested in sentences having negation word as Q_1 or Q_2 or the negation on the verb.

(5.1) $(S(NP(DET\ Q_1)(TP[p_1,...,p_m]X))(VP(v\ R)(NP(DET\ Q_2)(TP[q_1,...,q_n]Y))))$

In computing the meaning of a sentence, the presence of negation word as the determiner in a noun phrase is checked. Also the presence of negation on the verb during the parse of a sentence (i.e., presence of the word 'did not' before the verb) is checked. Example code is given for both the cases below with discussion on them in detail. In checking the presence of negation in a sentence the Boolean attributes `negationInNounPhrase` and `negationOnRelation` are set to either true or false. The Boolean value of the attribute `negationInNounPhrase` determines the presence or absence of the negation word 'No' as determiner in a noun phrase and the Boolean value of the attribute `negationOnRelation` determines the presence or absence of the negation on the verb (i.e., the word 'did not' before the verb).

These Boolean attributes are checked for computing the meaning of each noun phrase and then computing the meaning of verb phrase and finally computing the meaning of a sentence form the meaning of a noun phrase and a verb phrase.

```
_SENTENCE obj;
obj.meaning.checkNegation();

class LogicalForm extends Meaning
{       Relation                relation;
        QuantifiedConcept       qConcept1;
        QuantifiedConcept       qConcept2;
        boolean                 negationInNounPhrase;
...
        LogicalForm( QuantifiedConcept qc, Clause c )
        {       relation = c.relation;
                qConcept1 = qc;
                qConcept2 = c.qConcept;
        }
        public void checkNegation()
        { negationtInNounPhrase = qConcept1.checkForNegation() ||
                                        qConcept2.checkForNegation();
        }
...
}

class QuantifiedConcept extends Meaning
{       Concept                 concept;
        Quantifier              quantifier;

        QuantifiedConcept( Quantifier quant, Concept con )
        {       quantifier = quant;
                concept = con;
        }

        public boolean checkForNegation()
        {  if ( (quantifier.quantifier.toLowerCase()).comapreTo("no") == 0 )
                return true;
           else
                return false;
        }
}
```

```
class _TVERB extends LingObj
{       Relation        meaning = new Relation();
        ...
        public void parse()
        {
                negationOnVerb();
                ...
        }
        public void negationOnVerb()
        {
                if( (String)input.elementAt(0).compareTo("did not") == 0)
                meaning.checkNegation((String)input.elementAt(0));
                ...
        }
}
class Relation extends Meaning
{       boolean         negationOnRelation;
        ...
        public void checkNegation(String token)
        {
                if( token.compareTo("did not") == 0 )
                        negationOnRelation = true;
                else
                        negationOnRelation = false;
        }
}
```

The attributes `relation`, `qConcept1`, `qConcept2` in class LogicalForm and the attributes `concept`, `quantifier`, in class QuantifiedConcept are set to their respective meaning objects during the parse of a sentence. These attributes are used for checking the presence of negation word as determiner. The method `checkNegation()` in class `LogicalForm` sets the Boolean attribute `'negationInNounPhrase'` and the method `checkNegation()` in class Relation sets the Boolean attribute `'negationOnRelation'`. The basic idea here is to set these Boolean attributes before computing the meaning of a sentence. In computing it's meaning the meaning of each noun phrase and then the meaning of verb phrase are obtained by checking for the presence of negation as indicated by these Boolean attributes.

5.1.1 Example Sentence

Here we explain with an example the above-discussed details.

Consider the following examples.

(5.2) No man lifted a book.
(5.3) A man lifted no book.
(5.4) A man did not lift any book.

We discuss the details of setting the Boolean attributes `negationInNounPhrase` and `negationOnRelation` for example (5.2) in the figure (4).

Similarly,

For example (5.3)

`negationInNounPhrase` = true;

`negationOnRelation` = false;

For example (5.4)

`negationInNounPhrase` = false;

`negationOnRelation` = true;

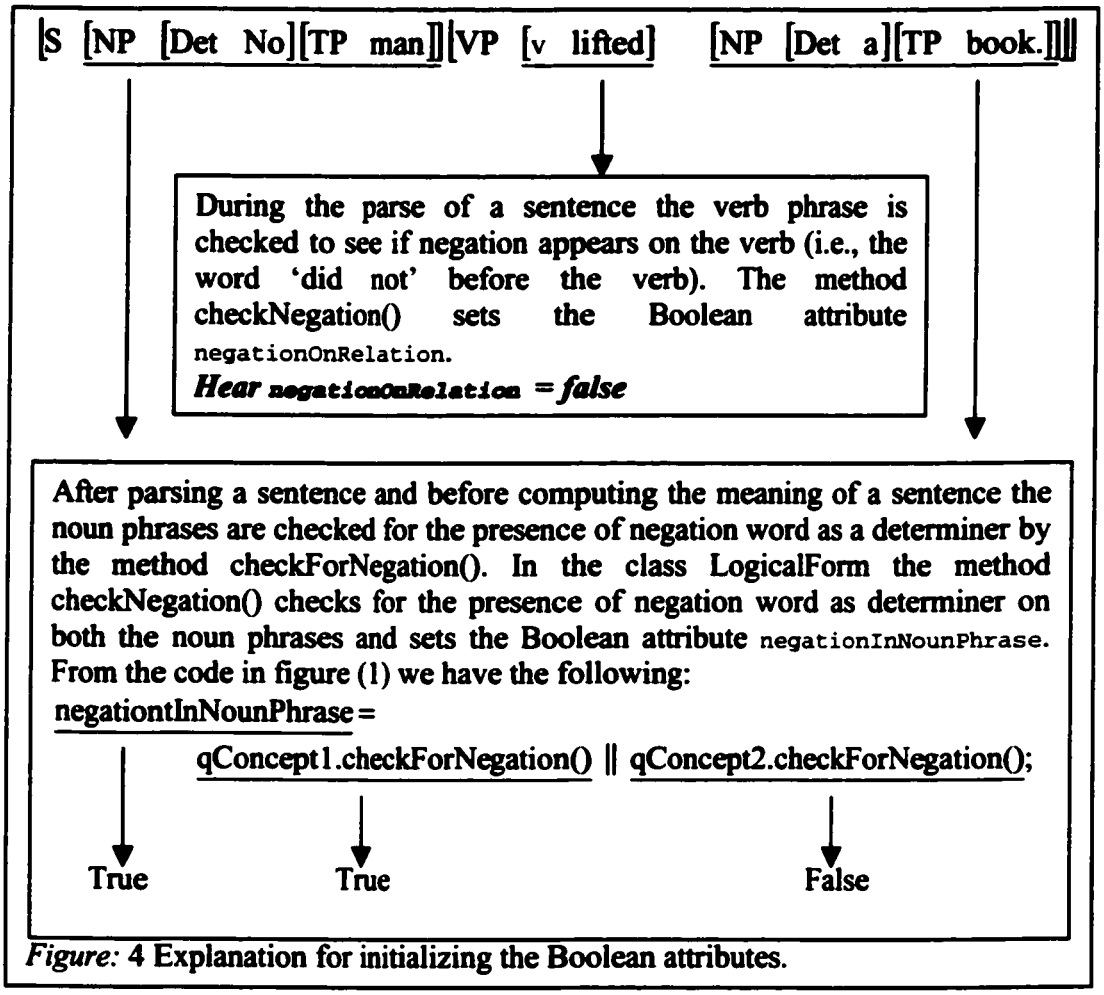


Figure: 4 Explanation for initializing the Boolean attributes.

5.2 Meaning of a Sentence

The meaning of a sentence is its logical form. A sentence involves two noun phrases of which one of them combines with a verb forming a verb phrase. Two logical forms (i.e., meanings) for the same sentence are obtained with each logical form having a different noun phrase as a head noun phrase. In chapter-4 the implementation details for obtaining meanings of a sentence was discussed.

This chapter discusses with an example, the details of obtaining the meanings of a sentence involving negation and shows some code.

Basically obtaining the meaning of a sentence can be explained as follows:

(5.5) $[S[NP_1][VP[_V R][NP_2]]]$

A sentence involves two noun phrases and a verb. Before computing the sentence meanings the Boolean attributes `negationOnRelation` and `negationInNounPhrase` are initialized. These attributes indicate the presence of negation on the verb or the negation word in the noun phrase while computing the meanings of a sentence. The meaning of a sentence is obtained by,

(Step: 1) Computing the meanings of each noun phrase (i.e., meaning of NP_1 and NP_2).

- Meaning of a noun phrase involving a quantifier could be either of the following

$$\text{Noun Phrase}(NP_1) \equiv \text{"Quantifier Concept"} = \text{FORALL}(X_1)(\text{Concept}(X_1) \rightarrow Q(X_1)) \text{ OR}$$
$$\text{Noun Phrase}(NP_1) \equiv \text{"Quantifier Concept"} = \text{EXISTS}(X_1)(\text{Concept}(X_1) \& Q(X_1))$$

$$\text{Noun Phrase}(NP_2) \equiv \text{"Quantifier Concept"} = \text{FORALL}(X_2)(\text{Concept}(X_2) \rightarrow Q(X_2)) \text{ OR}$$
$$\text{Noun Phrase}(NP_2) \equiv \text{"Quantifier Concept"} = \text{EXISTS}(X_2)(\text{Concept}(X_2) \& Q(X_2))$$

- Meaning of a noun phrase involving a negation word 'No' could be either of the following

$$\text{Noun Phrase}(NP_1) \equiv \text{"Negation Word 'No'Concept"} = \text{FORALL}(X_1)(\text{Concept}(X_1) \rightarrow \textit{not}\ Q1(X_1))$$
$$\text{Noun Phrase}(NP_2) \equiv \text{"Negation Word 'No'Concept"} = \text{FORALL}(X_2)(\text{Concept}(X_2) \rightarrow \textit{not}\ Q2(X_2))$$

where,
FORALL and EXISTS are quantifiers
X_1 and X_2 are variables
$\text{Concept}(X_1)$ and $Q(X_2)$ are predicates

The Concept can be modified (i.e., by nouns, adjectives, prepositional phrase, relative clause).

(Step: 2) Replacing the binary relation in the meaning of NP_2 thus obtaining the meaning of a verb phrase (i.e., predicate Q (X_2) in the meaning of the noun phrase NP_2 is replaced with the binary relation $_VR$ – relation between two concepts).

- Meaning of noun phrase (NP_2) is recomputed by replacing the predicate Q (X_2) with R (X_1, X_2). In case of negation on relation the predicate Q (X_2) is replaced with 'not R (X_1, X_2)'.
- The meaning of a verb phrase could be either of the following

Verb Phrase ≡ "Verb Quantifier Concept" = $FORALL(X_2)(Concept(X_2) \rightarrow R(X_1, X_2))$ OR
Verb Phrase ≡ "Verb Quantifier Concept" = $EXISTS(X_2)(Concept(X_2) \& R(X_1, X_2))$ OR
Verb Phrase ≡ "Verb Negation Word 'No'Concept" = $FORALL(X_2)(Concept(X_2) \rightarrow$ *not* $R(X_1, X_2))$

(Step: 3) The meaning of the noun phrase (NP_1) is recomputed thus obtaining the meaning of a sentence (i.e., the predicate Q1 (X_1) in the meaning of noun phrase NP_1 is replaced by the meaning of the VP – verb phrase).

- Meaning of a sentence could be one of the following:

Sentence ≡ "Noun Phrase (with a quantifier) Verb Noun Phrase (with a quantifier)" ≡
$EXISTS(X_1)(Concept(X_1) \& EXISTS(X_2)(Concept(X_2) \& R(X_1, X_2)))$ OR
$EXISTS(X_1)(Concept(X_1) \& FORALL(X_2)(Concept(X_2) \rightarrow R(X_1, X_2)))$ OR
$FORALL(X_1)(Concept(X_1) \rightarrow EXISTS(X_2)(Concept(X_2) \& R(X_1, X_2)))$ OR
$FORALL(X_1)(Concept(X_1) \rightarrow FORALL(X_2)(Concept(X_2) \rightarrow R(X_1, X_2)))$

Sentence ≡ "Noun Phrase (with a Quantifier) Verb Noun Phrase (with negation word 'No')" ≡
$EXISTS(X_1)(Concept(X_1) \& FORALL(X_2)(Concept(X_2) \rightarrow$ not $R(X_1, X_2)))$ OR
$FORALL(X_1)(Concept(X_1) \rightarrow FORALL(X_2)(Concept(X_2) \rightarrow$ not $R(X_1, X_2)))$

Sentence ≡ "Noun Phrase (with negation word 'No') Verb Noun Phrase (with a Quantifier)" ≡
$FORALL(X_1)(Concept(X_1) \rightarrow FORALL(X_2)(Concept(X_2) \rightarrow$ not $R(X_1, X_2)))$ OR
$FORALL(X_1)(Concept(X_1) \rightarrow EXISTS(X_2)(Concept(X_2) \&$ not $R(X_1, X_2)))$

Sentence ≡ "Noun Phrase (with a quantifier) Verb (negation on verb) Noun Phrase (with a quantifier)" ≡
$EXISTS(X_1)(Concept(X_1) \& EXISTS(X_2)(Concept(X_2) \&$ *did not* $R(X_1, X_2)))$ OR
$EXISTS(X_1)(Concept(X_1) \& FORALL(X_2)(Concept(X_2) \rightarrow$ *did not* $R(X_1, X_2)))$ OR
$FORALL(X_1)(Concept(X_1) \rightarrow EXISTS(X_2)(Concept(X_2) \&$ *did not* $R(X_1, X_2)))$ OR
$FORALL(X_1)(Concept(X_1) \rightarrow FORALL(X_2)(Concept(X_2) \rightarrow$ *did not* $R(X_1, X_2)))$

(Step: 3) The noun phrases are swapped and the above steps are repeated thus obtaining

the meanings of a sentence.

The table below shows the determiners with their replacement in the meaning of a syntactic constituent where they occur. The main point in sketching the table below indicate that the meanings of a sentence considered are those whose determiners can be replaced by either FORALL or EXISTS only. This is to avoid any confusion in explaining the meaning of a sentence by limiting our self to these determiners in the explanation.

Determiner	Replaced By	Determiner	Replaced By
No	FORALL	One or 1	EXISTS
Every	FORALL	Two or 2	E! (2:)
Each	FORALL	Three or 3	E! (3:)
All	FORALL	Four or 4	E! (4:)
Some	EXISTS	Five or 5	E! (5:)
A	EXISTS	...	...
An	EXISTS	...	...
Few	E!(few)	...	...
Many	E!(many)	...	...
Most	E!(most)	...	...
...	...	...	...

Table: 9 Determiners with their replacement in the meaning of a syntactic constituent where they occur.

The following figure shows with an example the above-discussed steps. Each syntactic constituent is labeled with a step number discussed above and number, which indicates relevant method of a class computing the meaning of that syntactic constituent. Consider the example with negation word in a noun phrase:

No man lifted a book.

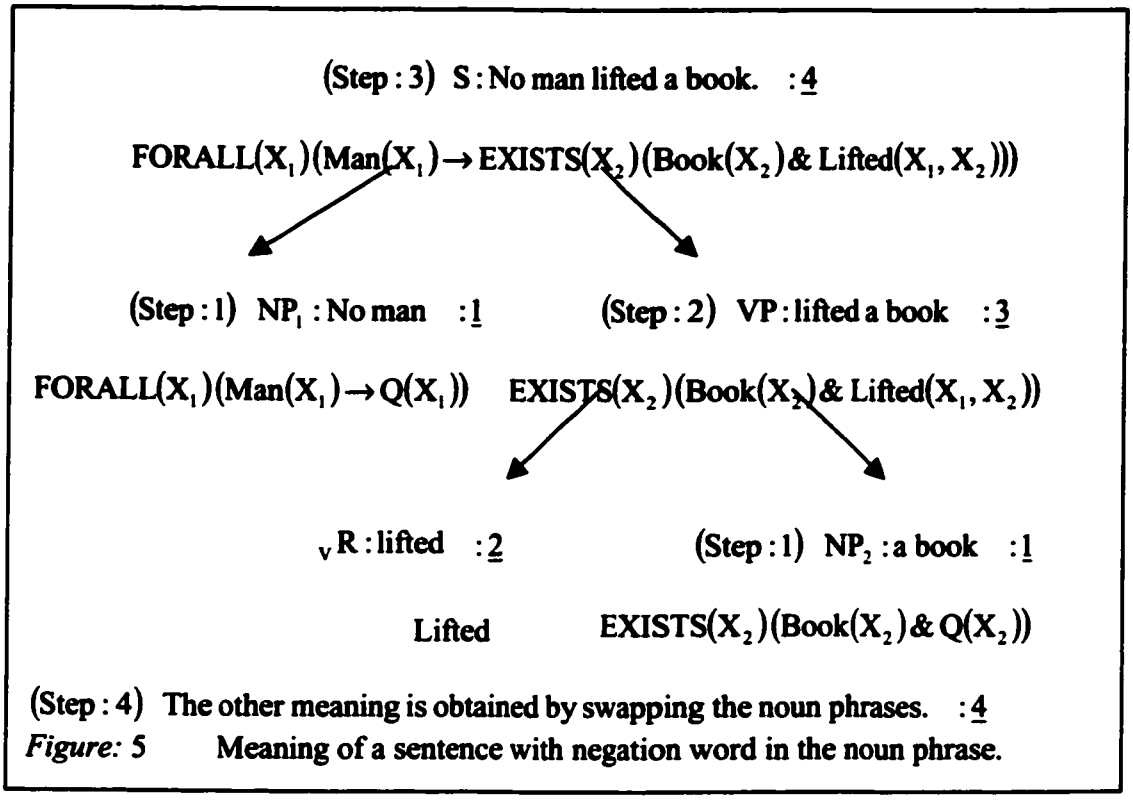


Figure: 5 Meaning of a sentence with negation word in the noun phrase.

A man did not lift any book.

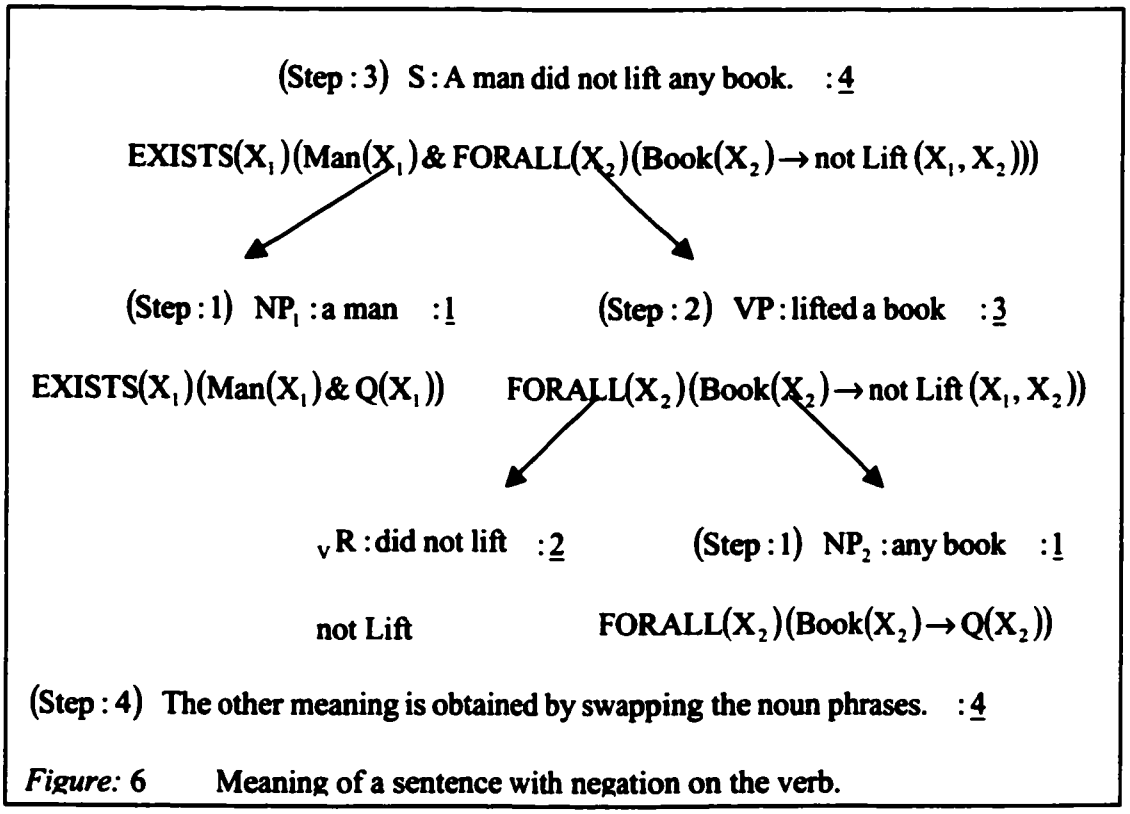


Figure: 6 Meaning of a sentence with negation on the verb.

Notations Used in Figures (6) And (7)	Methods of a class used in computing the meaning of a syntactic constituent.
:1	Public Expression1 getExpression(int varCount, boolean negationInNounPhrase) & Public void makeExpression(int varCount, boolean negationInNounPhrase) In class QuantifiedConcept extends Meaning Expression1(int vCount, Quantifier quant, Concept concept, boolean negationInNounPhrase) In Class Expression1 extends Expression
:2	
:3	Public void replaceBinaryRelation(String rel, boolean negationOnRelation) In Class Expression1 extends Expression
:4	Public String getScopeNeutralLF() & Public String getLogicalForm(QuantifiedConcept qConcept1, QuantifiedConcept qConcept2) In class LogicalForm extends Meaning Public void applyExpression(Expression1 expr, boolean negationInNounPhrase) & Public String getMatrix() In Class Expression1 extends Expression

Table: 10 Class methods used in obtaining the meaning of a syntactic constituent.

The numbers indicate labels in figure 6 and figure 7.

5.3 Example Sentence with Relevant Code Details

This section presents code and discusses some methods of a class mentioned in table 2. These methods form a meaning for a syntactic constituent. Chapter–4 discussed the attributes defined in each of these classes. Consider the sentence (5.5) and example (5.2) repeated below,

(5.5) $[S[NP_1][VP[_V R][NP_2]]]$

(5.2) No man lifted a book.

The meanings of a sentence indicate both direct and indirect readings. The meaning of the direct or indirect readings is obtained by,

- First computing the meanings each noun phrase (NP_1 and NP_2),

- Second the meaning of verb phrase (VP) is computed from the meaning of the verb ($_{V}R$) and the meaning of the noun phrase (NP_2),
- Third the meaning of a reading is computed form the meaning of the noun phrase (NP_1) and meaning of verb phrase (VP).

5.3.1 Meaning of Noun Phrase

For a sentence (5.5) the meaning of each noun phrase (NP_1 and NP_2) is computed by the methods `getExpression(int varCount, boolean negationInNounPhrase)` and `makeExpression( int varCount, boolean negationInNounPhrase)`. These methods initialize the attributes of Class Expression1 that form the meaning of a noun phrase. The meaning of a noun phrase can be retrieved by the method getMatrix() in Class Expression1 which is explained in chapter-4. That is, for the example of (5.2) these methods initializes the attributes of Class Expression1 shown in table (11).

No man lifted a book.

Attributes of Class Expression1	Attributes initialized for meaning of the noun phrase NP_1	Attributes initialized for meaning of the noun phrase NP_2
Quantifier	FORALL	EXISTS
Variable	(X1)	(X2)
Predicate1	MAN (X1)	BOOK (X2)
Connective	→	&
Predicate2	not Q1 (X1)	Q2 (X2)

Table: 11 Attributes of Class Expression1 initialized for meaning of noun phrases in a sentence.

The meaning of a noun phrase NP_1 is $\text{FORALL}(X1)(\text{MAN}(X1) \rightarrow \text{not } Q1(X1))$

The meaning of a noun phrase NP_2 is $\text{EXISTS}(X2)(\text{BOOK}(X2) \& Q2(X2))$

The noun phrase NP_1 (i.e., No man) has the negation word 'No'. The constructor in the Class Expression1 checks Boolean attribute negationInNounPhrase and the quantifier of the noun phrase thus initializing the attributes of this class as shown in the table (3). The code for these methods is given below. The arrow mark indicates a comment.

```
class QuantifiedConcept extends Meaning
{
      Concept             concept;
      Expression1         expression1;
      Quantifier          quantifier;

   public Expression1 getExpression(int varCount, boolean negationInNounPhrase)
      {     Expression1 expr = new Expression1(varCount,quantifier,concept,
                                                   negationInNounPhrase);
            makeExpression(varCount, negationInNounPhrase);
            expr.quantifier = expression1.quantifier;
            expr.variable = expression1.variable;
            expr.predicate1 = expression1.predicate1;
            expr.predicate2 = expression1.predicate2;
            expr.connective = expression1.connective;
            return expr;
      }
```

This method initializes the attributes of Class Expression1 for the meaning a noun phrase (i.e., NP_1 or NP_2)

```
      public void makeExpression( int varCount, boolean negationInNounPhrase)
      {
            if ( quantifier != null )
            {     Vector allAdjs = (concept.aModifiers).adjectives;
                  expression1 = new Expression1( varCount, quantifier,
                                          concept, negationFlag );
            // Update the predicates based on Noun Modifiers
                  for ( int i=(concept.nModifiers).size()-1; i>=0; i-- )
                  {     Concept oneModConcept = (Concept)
                              ((concept.nModifiers) .elementAt(i));
                  expression1.addAConceptConjunctToPredicate(oneModConcept );
            }
            // Update the predicates based on Adjectives Modifiers
                  for ( int i=allAdjs.size()-1; i>=0; i-- )
                  {     String oneAdj = (String)allAdjs.elementAt(i);
                        expression1.addAnAdjConjunctToPredicate( oneAdj );
                  }
            }
            else
            {     expression2 = new Expression2( concept );
            }
      }
}
```

This method updates the attribute predicate1 of Class Expression1 based on noun modifiers and adjective modifiers. Considering the example (5.2) the attribute predicate1 is MAN (X1) for NP_1 and BOOK (X2) for NP_2.

```
{
        int             varCount = 0;
        int             nextPredicate = 1;

        String quantifier = new String("");
        String variable = new String("");
        String predicate1 = new String("");
        String predicate2 = new String("");
        String connective = new String("");

        Expression1( int vCount, Quantifier quant, Concept concept, boolean
                                                        negationInNounPhrase )
        {
                varCount = vCount;
                variable = getVariable(varCount);
                quantifier = getQuantifier(quant.quantifier);
                connective = getConnective(quant.quantifier);
                predicate1 = getPredicate(concept);
                predicate2 = ((new String("Q")) + (new Integer(vCount+1)).
                                                        toString()) + variable;
                if(negationInNounPhrase & ((quant.quantifier.toLowerCase()).
                                                        CompareTo("no") == 0 ) )
                predicate2 = new String("not ").concat(predicate2);
                nextPredicate++;
        }
```

The constructor sets the attributes of the class. For the example of (5.2) values set for these attributes which form the meaning of a noun phrase (i.e., NP_1 or NP_2) are shown in table: 3.
A noun phrase may have negation word 'No'. The Boolean attribute `negationInNounPhrase` and the quantifier of the noun phrase are checked thus setting the attribute `predicate2`. For the example of (5.2) the Boolean attribute `negationInNounPhrase` is true for NP_1 and false for NP_2. Thus the attribute `predicate2` is set to 'not Q1 (X1)' for NP_1 and 'Q2 (X2)' for NP_2.

5.3.2 Meaning of Verb Phrase

For a sentence (5.5) the meaning of the verb phrase is computed by the method `replaceBinaryRelation( Expression1 expr, String rel, boolean negationOnRelation )`. Once the meaning of noun phrases NP_1 and NP_2 are computed, the meaning of verb phrase is obtained by updating the attribute `predicate2` of Class Expression1 in the meaning of noun phrase NP_2. That is in example (5.2), the meaning of noun phrase NP_2 for direct reading is $\text{EXISTS(X2)(Book(X2)\&Q2(X2))}$. Here predicate Q (X2) is

replaced by the relation Lifted thus forming the meaning of verb phrase $EXISTS(X2)(Book(X2) \& Lifted(X1, X2))$.

The code of this method is given below. The arrow mark indicates a comment.

```
class Expression1 extends Expression
{
        int             varCount = 0;
        int             nextPredicate = 1;

        String quantifier = new String("");
        String variable = new String("");
        String predicate1 = new String("");
        String predicate2 = new String("");
        String connective = new String("");
        String matrix = new String("");

        public void replaceBinaryRelation( Expression1 expr, String rel, boolean
                                                        negationOnRelation )
        {
                predicate2 = new String();
                predicate2 = rel + variable;
                if (negationOnRelation)
                        predicate2 = new String("not ").concat(predicate2);
                String oldPred2 = new String("");
                oldPred2 = predicate2;
                int len = oldPred2.length();
                oldPred2 = oldPred2.substring(0,len-3)
                        + expr.variable.substring(1,3) + ","
                        + oldPred2.substring(len-3);
                predicate2 = oldPred2;

        }
}
```

This method replaces the attribute `predicate2` in the meaning of NP_2 thus forming the meaning of verb phrase. Also the Boolean attribute `negationOnRelation` is checked for the presence of negation on verb.
That is for the example of (5.2) the meaning of NP2 is $EXISTS(X2)(Book(X2) \& Q2(X2))$, Q2(X2) is replaced by the binary relation thus forming the meaning of verb phrase $EXISTS(X2)(Book(X2) \& Lifted(X1, X2))$

5.3.3 Meaning of a Sentence

For a sentence (5.5) the meanings are computed by the methods `getScopeNeutralLF()`, `getLogicalForm( QuantifiedConcept qConcept1, QuantifiedConcept qConcept2 ) of Class LogicalForm` and `applyExpression( expr2a, negationPresent) of Class Expression1`. Once the meaning of noun phrase (NP_1) and the meaning of verb phrase (VP) are computed the meaning of a sentence is computed by replacing the attribute predicate2 for the meaning of NP_1 by the meaning of verb phrase (VP), thus obtaining the meaning of a sentence.

This is explained with an example.

No man lifted a book.

For the direct reading,

Meaning of noun phrase is $\text{FORALL}(X1)(\text{MAN}(X1) \rightarrow \text{not } Q1(X1))$.

Meaning of verb phrase is $\text{EXISTS}(X2)(\text{Book}(X2) \& \text{Lifted}(X1, X2))$.

The method `applyExpression (expr2a, negationInNounPhrase)` checks for the presence of negation word in the noun phrase NP_1 that is indicated by the Boolean attribute `negationInNounPhrase` (the Boolean attribute only specifies the presence of negation word in either of the noun phrases) and the presence of the word 'not' in the attribute predicate2 (this is shown in table 3, this is an indication for the presence of negation word in a specific noun phrase). For the above example the negation word is in the noun phrase (NP_1). In the presence of negation word the meaning of verb phrase is recomputed.

Meaning of verb phrase is $\text{FORALL}(X2)(\text{BOOK}(X2) \rightarrow \text{not Lifted}(X1, X2))$.

Meaning of sentence is (i.e., direct reading)

$\text{FORALL}(X1)(\text{MAN}(X1) \rightarrow \text{FORALL}(X2)(\text{BOOK}(X2) \rightarrow \text{not Lifted}(X1, X2)))$

The code for these methods is given below. The arrow mark indicates a comment.

```
class LogicalForm extends Meaning
{
      Relation            relation;
      QuantifiedConcept   qConcept1;
      QuantifiedConcept   qConcept2;
      boolean             negationInNounPhrase = false;

      public String getLogicalForm( QuantifiedConcept qConcept1,
                                    QuantifiedConcept qConcept2 )
      {
            Expression1 expr1a = qConcept1.getExpression(0, negationPresent);
            Expression1 expr2a = qConcept2.getExpression(1, negationPresent);
            expr2a.replaceBinaryRelation( relation.relation, negationPresent,
                              relation.checkNegationOnRelation());
            expr1a.applyExpression( expr2a, negationPresent);
            return expr1a.getMatrix();
      }

      public String getScopeNeutralLF()
      {     String finalRes = new String();
            finalRes = finalRes + "[\n";
            finalRes = finalRes + "\t"
                              +     getLogicalForm(qConcept1,qConcept2)      +
";\n\t"
                              + getLogicalForm(qConcept2,qConcept1);
            finalRes = finalRes + "\n]\n";

            return finalRes;
      }
}
```

In the method getScopeNeutralLF() the noun phrases are swapped for a sentence $[S[NP_1][VP[_V R][NP_2]]]$ to obtain its meanings.

The method getLogicalForm(QuantifiedConcept qConcept1, QuantifiedConcept qConcept2) obtains the meaning of a sentence $[S[NP_1][VP[_V R][NP_2]]]$ that is

$$FORALL(X_1)(Man(X_1) \rightarrow EXISTS(X_2)(Book(X_2) \& Lifted(X_1, X_2))) \ \&$$
$$EXISTS(X_1)(Book(X_1) \& FORALL(X_2)(Man(X_2) \rightarrow Lifted(X_1, X_2)))$$

```
class Expression1 extends Expression
{
       int          varCount = 0;
       int          nextPredicate = 1;

       String quantifier = new String("");
       String variable = new String("");
       String predicate1 = new String("");
       String predicate2 = new String("");
       String connective = new String("");

public void applyExpression( Expression1 expr, boolean negationFlag )
       {
         if (negationFlag)
         {
            String negationInExpr1 = new String(predicate2.substring(0,3));
            String negationInExpr2 = new String(expr.predicate2. substring
                                                                  (0,3));
            if ( negationInExpr1.compareTo("not") == 0 )
            {
               if ( (expr.quantifier.toUpperCase()).compareTo("FORALL") == 0 )
               {
                    expr.quantifier = new String("EXISTS");
                    expr.connective = "&";
                }
                else if ( (expr.quantifier.toUpperCase()).compareTo("EXISTS")
                                                                       == 0 )
               {
                    expr.quantifier = new String("FORALL");
                    expr.connective = "->";
                }
                else if ( negationInExpr2.compareTo("not") == 0 )
                 {

                 }
            }
         }

        if ( nextPredicate == 1 ) predicate1 = expr.getMatrix();
             else predicate2 = expr.getMatrix();
       }
}
```

Chapter 6

Conclusion

6.1 Extending quantifier scope resolution algorithm

In this thesis, an extension to the QC algorithm of Saba&Corriveau was described. The purpose of this extension was to investigate the thesis that the QC algorithm of Saba&Corriveau can be extended to account for negation.

The extension is formulated by assigning meaning to the negation word 'No' (i.e., $\lambda P \lambda Q [(\forall x)(P(x) \rightarrow Q(x))]$) as in (Montague, 1973) thus pushing negation on to the verb for a sentence of the form

$$(S(NP(DET\ Q_1)(TP[p_1,...,p_m]X))(VP(v\ R)(NP(DET\ Q_2)(TP[q_1,...,q_n]Y)))).$$

This sentence may involve negation word 'No' in the determiner position (i.e., Q_1 or Q_2) and quantifiers like a, every, all, any in the other determiner position (i.e., Q_1 or Q_2) **and** also negation may occur on the verb. Various QC's for sentences are considered.

The following is a summary of the work that has been done:

1. An extension to the QC algorithm of Saba, 1999 is tabulated.
2. The QC algorithm can be readily extended to account for negation.

   For a sentence,

   $$(S(NP(DET\ Q_1)(TP[p_1,...,p_m]X))(VP(v\ R)(NP(DET\ Q_2)(TP[q_1,...,q_n]Y)))).$$
3. Negation word like 'No' in the determiner position (i.e., Q_1 or Q_2) with quantifiers like a, every, all, any in the other determiner position (i.e., Q_1 or Q_2) for a sentence is accounted by the tabulated results.
4. Negation on the verb (i.e., $\neg\ _vR$) is also accounted by the tabulated results for a sentence with

Q_1 = a & Q_2 = any or all **and**

Q_1 = every & Q_2 = any.

5. Negation at the linguistic level can be manipulated at the logical form level, and can always be translated into a unary logical operator.
6. The QC algorithm can then be applied as before, but in some cases it seems that negation require one to reverse the predictions of the QC algorithm.
7. Analyzed all patterns and generated the rules required to extend the QC algorithm.

6.2 Analysis of the Experiment

This report described a natural language interpreter for resolution of quantifier scope ambiguity that was built to test the following thesis:

The QC algorithm can be readily extended to account for negation.

We have demonstrated that such an extension is possible. From the meaning of a sentence we see that negation is pushed on to the verb. Thus in place of determiners we have quantifiers like a, all, every, any, etc.,

The conclusions drawn form the experiment can be summarized as follows.

1. The QC algorithm can be readily extended to account for negation.
2. Negation at the linguistic level is manipulated at the logical form level, and is translated into a unary logical operator (this is represented by 'not' before the verb in the experiment).
3. The QC algorithm is applied as before, but in some cases where negation occurs the predictions of QC algorithm are reversed.

Chapter 7

Future Work

There are other linguistic forms of negation that could also be analyzed. That is for a sentence $Q_1\,[p_1, p_2, \ldots, p_n]C_1\ R\ Q_2\,[q_1, q_2, \ldots, q_m]C_2$ 'not-all' can appear as a determiner.

Example Sentence:

Not all graduate students attended $\left\{\begin{matrix} a \\ every \\ all \end{matrix}\right\}$ seminar(s) on KDD.

Negation combined with other generalized quantifiers could also be analyzed. That is for a sentence $Q_1\,[p_1, p_2, \ldots, p_n]C_1\ R\ Q_2\,[q_1, q_2, \ldots, q_m]C_2$ 'most, but not-all' can appear as a determiner.

Example Sentence:

Most, but not all programmers that work for a text retrieval company in Ottawa visited MIT.

The exact interplay between negation and the passive/active transformation as well as negation and branching quantifiers could also be studied further.

Bibliography

Allen, J., 1987, 'Natural Language Understanding', Benjamin Cummings Publishing Company, Inc. Menlo Park, California.

Alshawi, H., 1990, 'Resolving quasi logical forms', Computational Linguistics, 16: 133-134.

Bibel, W., & Jorrand, Ph., editors 1986, 'Preface', Fundamentals of Articial Intelligence, volume 232 of Lecture Notes in Computer Science, Springer-Verlag, Berlin.

Cherniak, C., 1992, 'Minimal Rationality', MIT Press, Cambridge.

Corriveau, J-P., (1995), 'Time-Constrained Memory: A Reader-Based Approach to Text Comprehension', Lawrence Erlbaum Associates, NJ.

Cresswell, M.J., 1973, 'Logics and Languages', Methuen & Co Ltd., London.

Gamut, L. T. F., 1991, 'Logic Language and Meaning – Volume 2: Intensional Logic and Logical Grammar', University Of Chicago Press, Chicago.

Geach., (1968), 'Reference and Generality', Cornell University Press, Ithaca, New York.

Geurts, B., 2002, 'Donkey Business', Linguistics and Philosophy - A Journal of Natural Language Syntax, Semantics, Logic, Pragmatics and Processing. 25(2): 129-156.

Grosz, B., Appelt, D., Martin, P., and Pereira, F. C. N., 1987, 'TEAM: An Experiment in

the Design of Transportable Natural-Language Inferences', In Artificial intelligence, 32: 173-243.

Hobbs, J.R., 1996, 'Monotone decreasing quantifiers in a scope-free logical form', in K. van Deemter and S. Peters, eds., Semantic Ambiguity and Underspecification, CSLI publications, Stanford, CA, pp. 55-76.

Hoobs, J.R., & Shieber, S.M., 1987, 'An algorithm for generating quantifier scooping', Computational Lingustics, 13: (1&2), 47-55.

Irene, H., 1990, 'E-Type Pronouns and Donkey Anaphora', Linguistics and Philosophy – A Journal of Natural Language Syntax, Semantics, Logic, Pragmatics and Processing, 13: 137-177.

Johnson-Laird, P. N., 1994, 'Mental Models and Probabilistic Thinking, Cognition, 50: 198-209.

Kurtzman, H., & Macdonald, M., 1993, 'Resolution of quantifier scope ambiguities', Cognition – International Journal of Cognitive Science, 48: 243-279.

Minsky, M., 2000, 'Commonsense-Based Interfaces', In Communications of the ACM, 43(8): 66-73.

May, R., 1989, 'Interpreting logical form', Linguistics and Philosophy, 12(4): 387-436.

May, R., 1985, 'Logical Form: Its Structure and Derivation', MIT Press, Cambridge,

Mass.

McCarthy, J., 2001, 'What is Artificial Intelligence?', Revised September 28, 2001.

McCarthy, J., 1989, 'Artificial Intelligence, Logic and Formalizing Commonsense', In Richard Thomason, eds., Philosophical Logic and Artificial Intelligence, Kluwer Academic Publishers, Dordrecht, Netherlands, pp. 161-190.

Montague, R., 1973, 'The proper treatment of quantification in ordinary English', In J. Hintikka, J. Moravcsik, and P. Suppes eds., Approaches to language, pp. 221-242. Dordrecht: D. Reidel. Reprinted in Montague (1974, 247-270).

Montague, R., 1974, 'Proper treatment of quantification in natural language', In R. Thomason, eds., Formal Philosophy: Selected Papers of Richard Montague, CT: Yale University Press, New Haven.

Moran, D., 1988, 'Quantifier scoping in the SRI core language engine', In Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics – ACL '88, pp. 33-40.

Moran, D., and Pereira, F.C.N., 1992, 'Quantifier Scope', In H. Alshawi eds., The Core Language Engine, MIT Press, Cambridge, MA.

Park, J., 1995, 'Quantifier scope and constituency', In Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics - ACL '95, ACL/MIT Press, pp. 205-212.

Partee, B., 1974, 'Opacity and Scope', In M.K. Munitz and P. K. Unger, eds., Semantics and Philosophy, New York University Press, New York. pp. 81-101.

Poesi, M., 1996, 'Semantic Ambiguity and Perceived Ambiguity, In K. van Deemter and S. Peters, eds., Semantic Ambiguity and Underspecification, CSLI Publications, Stanford. pp. 159-201.

Pore, E., & Garson, J., 1983, 'Pronouns and Quantifier-Scope in English', Journal of Philosophical Logic, 12: 327-358.

Reinhart, T., 1997, 'Quantifier Scope: How labor is divided between QR and choice functions', Linguistics and Philosophy - A Journal of Natural Language Syntax, Semantics, Logic, Pragmatics and Processing, 20(4): 335-397.

Reinhart, T., 1983, 'Anaphora and Semantic Interpretation', Chicago University Press, Croom-Helm.

Saba, W. S., and Corriveau, J.-P., 1995, 'Context, Quantification, and Cognitive Plausibility', In Proceedings of IJCAI-95 Workshop on Context in Natural Language, Montreal, pp. 121-128.

Saba, W. S., 1995, 'Towards a Cognitively Plausible Model for Quantification', In Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics – ACL '95, Cambridge, MA, pp. 223-225.

Saba, W. S., & Corriveau, J.-P., 1997, 'A pragmatic treatment of quantification in natural language', In Proceedings of the 14th National Conference of the American Association for Artificial Intelligence – AAAI '97, RI, USA. pp. 610-615.

Saba, W. S., and Corriveau, J.-P., 1999, 'Plausible Reasoning in NLP without Massive Amount of Background Knowledge', In Proceedings of the 3rd Australian Workshop on Commonsense Reasoning, pp. 144-159.

Saba, W. S. 1999, 'An Inferencing Strategy for Resolving Quantifier Scope Ambiguities', Ph.D. Thesis, School of Computer Science, Carleton University, Ottawa, Canada.

Saba, W. S., and Corriveau, J.-P., 2001, 'Plausible reasoning and the resolution of quantifier scope ambiguities', Studia Logica - International Journal of Symbolic Logic Special Issue on Commonsense Reasoning), 67: 271-289.

Scha, R., 1981, 'Distributive, Collective and Cumulative Quantification', In J. Groendijk et al. Eds, Formal Methods in the Study of Language (Part I & Part II), Mathematical Tracts, Amsterdam, pp. 483-512.

Shastri, L., and Ajjanagadda, V., 1993, 'From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic binding using temporal synchrony', Behavioral and Brain Science, 16: 417-494.

Sher, G., 1990, 'Ways of Branching Quantifiers', Linguistics and Philosophy, 13(4): 393-422.

Steedman, M., 1999, 'Alternating Quantifier Scope in CCG', In Proceedings of 37th Annual Meeting of the Association for Computational Linguistics – ACL' 99, College Park, Maryland, pp. 301-308.

Van Der Does, J., 1994, 'On Complex Plural Noun Phrases', In Kanazawa et al. Eds, 'Dynamics, Polarity, and Quantification', CLSI Publication, Stanford, CA, pp. 81- 118.

Appendix-A Ambiguityes in Natural Language and Problems in Logical Form

Some ambiguityes in Natural language pose problems in the logical form. Various ambiguityes in natural language and problems faced in the logical form are discussed below. The common theme form the discussion is that some of these ambiguityes can be resolved using commonsence knowledge.

1) Prepositional phrase attachment
2) Wrong interpretation in logical from
3) Unbound variable problem
4) Opaque context
5) Reference resolution
6) Quantifier scope ambiguity

A.1 Prepositional Phrase Attachment

A parse for natural language must often choose between two or more equally grammatical parses for the same sentence. This arises in prepositional phrase attachment, which is a common cause of structural ambiguity in natural language. That is the prepositional phrase can attach either to the noun phrase or the verb phrase that are equally grammatical thus giving two or more meanings for the same sentence. The following example is of interest as it involves a quantifier in the prepositional phrase thus giving rise to scoped readings on interaction with the noun phrase.

For example:

(1) (2) John $\left\{ \begin{matrix} \text{visited} \\ {}_{(VP)}\text{advertised} \end{matrix} \right\} [{}_{(NP)}\text{a house}][{}_{(PP)}\text{on every street}]$

In (1) the prepositional phrase '*on every street*' can attach to either a NP '*a house*' or VP '*visited*' as in (1.1) and (1.2) respectively.

(1.1) ⇒ John [(VP) Visited]
((NP) a house
((PP) on every street))

(1.2) ⇒ John [(VP) Visited]
((PP) on every street)
((NP) a house)

In (1.1) we are speaking of the same house on every street, that is visited, and so this interaction is wrong. Where as in (1.2) we are speaking of some house (not necessarily the same), which is visited. In this case VP attachment is the correct interpretation.

In (2) the prepositional phrase '*on every street*' can attach to either a NP '*a house*' or VP '*visited*' as in (2.1) and (2.2) respectively.

(2.1) ⇒ John [(VP) Advertised]
((NP) a house
((PP) on every street))

(2.2) ⇒ John [(VP) Advertised]
((PP) on every street)
((NP) a house)

The cases in (1) and (2) differ, that is in (2.1) the same house can be advertised on every street and in (2.2) on every street some or a different house can be advertised.

From examples (1) and (2) we see that the attachment of prepositional phrase involving a quantifier requires world knowledge to disambiguate.

Other Examples:

(3) The agent $\left\{\begin{matrix}[_{(VP)}\ \text{saw}\ ] \\ [_{(VP)}\ \text{killed}]\end{matrix}\right\}[_{(NP)}\ \text{a criminal}][_{(PP)}\ \text{on every air plane.}]$

(4) John $[_{(VP)}\text{spoke}][_{(NP)}\text{at a conference}][_{(PP)}\text{on genetic algorithms}]$

A.2 Wrong Interpretation in Logical Form

Donkey sentences show cause of a problem in logical approach known as wrong interpretation in logical form.

To illustrate this consider the following example:

(5) If Pedro owns a donkey he beats it.

If we assume that the "if-then" construction of English is be translated as an implication then the representation of (5) is as shown in (6):

(6) $(\exists x)(\text{Donkey}(x) \wedge \text{Owns}(\text{Perdo}, x)) \rightarrow \text{Beat}(\text{Perdo}, x)$

This of course is not a logical sentence because the variable "X" in "Beat (Pedro, X)" is out side the scope of the existential quantifier and remains unbound. The only representation we can get for (5) in first order logic is as follows which is correct interpretation:

(7) $(\forall x)((\text{Donkey}(x) \wedge \text{Owns}(\text{Perdo}, x)) \rightarrow \text{Beats}(\text{Perdo}, x))$

If 'it' is bound, then wrong semantical rule for the quantifier is selected, whereas if the right rule is selected, 'it' is left unbound (Ernest Le Pore and James Garson, 1983).

The scope neural logical form mentioned in Saba and Corriveau (2000) solves this problem. 'if' is treated as a logical connective, although in the surface structure it is a sentential connective.

$\alpha =$ $(\exists s_1, s_2)$ (unique(s_1, {x / Perdo = x}) Λ a(s_2, {x / Donkey(x)})
Λ $(\forall y, z)((y \in s_1) \Lambda (z \in s_2)$
$\supset$ Own(y, z)))

$\beta =$ $(\exists s_1, s_2)$ (he(s_1, {x / REF}) Λ it(s_2, {x / REF})
Λ $(\forall y, z)((y \in s_1) \Lambda (z \in s_2)$
$\supset$ Beat(y, z)))

$\alpha \rightarrow \beta$

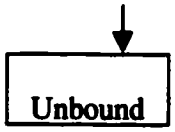
This form evaluates to true only if Perdo beats all the donkeys that he owns which is the correct reading of the sentence.

A.3 Unbound Variable Problem

Geach (1968) brought to modern attention the problem of unbound variable. Consider the following famous donkey example.

(8) Every farmer who owns a donkey beats it.

$(\forall x)(\text{Farmer}(x) \Lambda (\exists y)(\text{Donkey}(y) \Lambda \text{Own}(x, y)) \rightarrow \text{Beat}(x, y))$



A reader of this sentence assumes that 'it' is a bound variable co-referencing to an existential quantifier 'a donkey', which in turn ranges over a set of values as it is with in the scope of the universally quantified NP 'every farmer'. But this is not the case the pronoun *'it'* does not fall with in the scope of the NP to which it is bound (Allen J, 1995; Irene H, 1990; Steedman M, 1999).

It is generally acknowledged that donkey anaphora eludes any purely syntactic treatment of anaphoric binding, and that a pragmatic analysis is called for (Geurts B, 2000).

A.4 Opaque Context

There are two views of how opacity arises (Partee B, 1974). First view is that opacity results from the embedding of a sentence into a construction headed by a modal verb, adjective or adverb, such as necessary, believes that, must or certain. The second view as suggested by the work of Montague, is that opacity is one aspect of intensionality.

For Example:

(9) John believes that a republican will win the presidency.

The two possible readings of this sentence are shown below and pose a challenge to logical form. That is such sentences, which give rise to opaque context and involve quantifier scope give rise to scope orderings that have different truth conditions. The primary cause being the decision of weather belief falls under the scope of an identifier or identifier falls under the scope of belief.

(9.1 – Opaque Reading) $\text{Believes}(\text{John}, \exists r(\text{republican}(r) \wedge \text{WillWin}(r, \text{Presidency})))$

(9.2 – Non Opaque Reading) $\exists r(\text{republican}(r) \wedge \text{Believes}(\text{John}, \text{WillWin}(r, \text{Presidency})))$

In (9.1) "a republican will win the presidency" is in the scope of belief. Whereas in (9.2) what is believed is in the scope of "a republican". Roughly what (9.1) and (9.2) say are the following.

1 ⇒ On the de dicto reading: - john believes that some republican (i.e., no one in particular) will win the presidency.

2 ⇒ On the de re reading: - there is some (particular) republican that john believes will win the presidency.

A.5 Reference Resolution

With the advent of computers as a tool for man machine interaction the quality and robustness of natural language understanding systems has become critical. A major obstacle in building robust systems that interpret, extract information, summarize and answer questions from texts is the need to identify the entities referred to by pronouns or other referential expressions from texts.

For Example:

(10) John shot a policeman. He immediately

(a) fled away

(*b*) fell down

The pronoun 'he' can either refer to 'john' or to 'policeman'. In the case of the sentence continued with fled away 'he' most likely will refer to 'john' and in the case of the sentence continued with fell down 'he' most likely will refer to 'policeman'. Only a commonsense inference can help us to determine the most plausible of all that is possible. (Saba and Corriveau, 2001).

A.6 Scope Ambiguity

Scope ambiguity occurs with quantifiers, logical operators, modal operators, and adverbials in sentences. The scoping associated to a quantifier is determined by its interaction with other quantifiers, logical operators (e.g., modals and negation), and boundaries of certain syntactic constituents (e.g., major clauses and relative clauses) (Allen, 1987). This can be illustrated by a series of sentences that are ambiguous due to scope phenomena.

Quantifier scope ambiguity:

Any sentence that contains two or more quantifiers will likely be ambiguous. This ambiguity is caused due to the interaction of quantifiers in the sentence thus giving rise number of scope orderings. Typically a sentence with n quantifiers gives rise to n! scope orderings.

For example:

Every book is placed on a shelf.

This sentence describes two different readings: (1) every book is placed on some (different) shelf (2) a single shelf has every book placed on it.

Ambiguity arising from the scoping of logical operators and quantifiers: -

Example1:

All that glitters isn't gold.

Here the scope of the word not is unclear. This word could have either a *narrow scope* or a wide *scope*, creating in two possible meanings:

All that glitters is non-gold.

Not all of that which glitters is gold.

The first interpretation is a *narrow* scope of not, where the word "not" negates only the phrase "is gold." The second interpretation is a wide scope of "not" negates the entire sentence.

$(a)\forall x[\text{Glitter}(x) \rightarrow \neg \text{gold}(x)]$

$(b)\neg\forall x[\text{Glitter}(x) \rightarrow \text{gold}(x)]$

Example2:

John didn't enter every room.

Which may mean that john didn't enter any room, that is,

$(a)\forall x[\text{Room}(x) \rightarrow \neg\text{Enter}(\text{John}, x)]$

Or there is at least one room john didn't enter, that is,

$(b)\neg\forall x[\text{Room}(x) \rightarrow \text{Enter}(\text{John}, x)]$

Ambiguity with logical connectives: -

For example,

Everyone thought that king or queen would rule.

Which has one reading where everyone thought king would rule or everyone thought queen would rule, and another reading where each person thought that either king or queen might rule. The different readings here are caused due of the scoping between the universal quantifier and the disjunction.

Ambiguity due to tense operators and adverbials: -

Example,

He will feed the hungriest dog tomorrow.

This sentence may mean that the dog is hungriest now and will be fed later (possibly when it is not hungry), or that the dog will be the hungriest tomorrow when it is fed (and possibly is not hungry now).

Scope ambiguity on the adverbial: -

Example,

A clever player always wins the game.

The statement may mean that the winner is always a clever player or that there is a particular clever player who wins every game.

Conclusion

Some of the above processes are solvable purely logically (like, unbound variable problem, Wrong interpretation of logical form) and some others are solvable using commonsense knowledge (like Opaque context, Reference resolution, Quantifier scope ambiguity, Structural ambiguity where prepositional phrase interacts with a quantifier). Among these quantifier scope ambiguity is of interest.

Appendix-B Code For Some Methods

Below we give code for some methods as mentioned in chapter 5.

B.1 Code for terminal Linguistic Object

//A terminal linguistic object (i.e., Proper Noun) defined as a class.

// Note: terminal linguistic objects have only attribute for it's meaning.

```
class _PNOUN extends LingObj
{
	Concept		meaning = new Concept();

	_PNOUN( Vector words ) { super(words); }

	public void parse()
	{
		if ( input.size() > 0 )
		{
			String word = (String)input.elementAt(0);
			meaning = dictionary.getMeaningOfPNoun( word );

			if ( meaning != null )
			{
				parsedOK = true;
				numOfNodes = 1;
				parseTree = parseTree + "[PNOUN  " + word + "]";
				tokens.addElement( word );
				for ( int i=1; i<input.size(); i++ )
					restOfInput.addElement( input.elementAt(i) );
				meaning.constant = word;
			}
			else
			{
				parsedOK = false;
				restOfInput = input;
			}
		}
	}
};
```

B.2 Code for non-terminal Linguistic Object

// A non-terminal linguistic object (i.e., Verb Phrase) defined as a class.

// A non-terminal linguistic object has attributes for linguistic objects that is syntactic structure of the object at this syntactic level and meaning object that refers to the appropriate meaning class.

```
class _VP extends LingObj
{
	Clause		meaning;

	_TVERB		tverb  = new _TVERB(input);
	_NP			np		= new _NP(input);

	_VP( Vector words ) { super(words); }

	public void parse()
	{
		LingObj lObj = consistsOf( input, rule( tverb, np ) );

		parsedOK = lObj.parsedOK;
		restOfInput.removeAllElements();
		restOfInput = (Vector)(lObj.restOfInput).clone();
		tokens = lObj.tokens;
		parseTree = "[VP  " + lObj.parseTree + "]";
		numOfNodes = lObj.numOfNodes;

		computeMeaning();
	}

	// compositional semantic rules
	private void computeMeaning()
	{
		if ( parsedOK )
		{
			meaning = new Clause( tverb.meaning, np.meaning );
		}

		meaning.resolveAttachments();
	}

};
```

B.3 Code for Determining the Most Plausible Scope Ordering

```
class LogicalForm extends Meaning
{
      Relation            relation;
      QuantifiedConcept   qConcept1;
      QuantifiedConcept   qConcept2;
      Vector              readings = new Vector();

public void scope()
      {
            Vector reading1 = new Vector();
            Vector reading2 = new Vector();

            double r1 = scopeTop(qConcept1,qConcept2,1);
System.out.print("\nPLAUSIBILITY for direct reading = " + r1 );
            double r2 = scopeTop(qConcept2,qConcept1,0);
System.out.print("\nPLAUSIBILITY for indirect reading = " + r2 );

            if ( r1 >= r2 )
            {
                  reading1.addElement(new Double(r1));
                  reading1.addElement(getLogicalForm(qConcept1,qConcept2));
                  reading2.addElement(new Double(r2));
                  reading2.addElement(getLogicalForm(qConcept2,qConcept1));
System.out.print("\nDirect reading is more plausible.");
            }
            else
            {
System.out.print("\nIndirect reading is more plausible.");
                  reading1.addElement(new Double(r2));
                  reading1.addElement(getLogicalForm(qConcept2,qConcept1));
                  reading2.addElement(new Double(r1));
                  reading2.addElement(getLogicalForm(qConcept1,qConcept2));
            }

            readings.addElement( reading1 );
            readings.addElement( reading2 );
      }
```

B.4 Code for Obtaining the Parameters used in Measuring the Plausibility of a Scope Reading

```
public double scopeTop(    QuantifiedConcept quantCon1,
                           QuantifiedConcept quantCon2, int order )
      {
            Concept              concept1 = quantCon1.concept;
            Concept              concept2 = quantCon2.concept;
            Quantifier           quantifier1 = quantCon1.quantifier;
            Quantifier           quantifier2 = quantCon2.quantifier;

//System.out.print("\nAccounting for the linguistic context...");

      double       conceptX = quantCon1.getModifiersEffect();
      double       conceptY = quantCon2.getModifiersEffect();
      double       ratioA = conceptX/quantCon1.concept.typicalCardinality;
      double       ratioB = conceptY/quantCon2.concept.typicalCardinality;
      QC                  qc;
//System.out.print("\nRetrieving the relevant QC...");
      Vector       rangeV1; Vector            rangeV2;
      if ( order == 1 )
      {     qc = relation.getQC(quantCon1.concept,quantCon2.concept);
            rangeV1 = constructRange( qc.getFirst(), quantCon1.concept );
            rangeV2 = constructRange( qc.getSecond(), quantCon2.concept );
      }
      else
      {     qc = relation.getQC(quantCon2.concept,quantCon1.concept);
            rangeV1 = constructRange( qc.getSecond(), quantCon2.concept );
            rangeV2 = constructRange( qc.getFirst(), quantCon1.concept );
      }
//System.out.print("\nRetrieved the relevant QC " + qc.displayQC());
//System.out.print("\nConstructing appropriate ranges...");

Vector       rangeQ1=constructRange(quantifier1.quantifier,quantCon1.concept );
Vector       rangeQ2=constructRange(quantifier2.quantifier, quantCon2.concept);

//System.out.print("\nConstructed appropriate ranges...");
//System.out.print("\nMultiplying ranges by appropriate (linguistic context)
ratios...");
//System.out.print("\nDividing ranges...");

Vector       finalRange1 = divideRanges(rangeQ2,multiplyRange(ratioA,rangeQ1));
Vector       finalRange2 = divideRanges(rangeV2,rangeV1);

//System.out.print("\nComputing plausibility measure...");

            return computePlausibilityMeasure( finalRange1, finalRange2 );
      }
```

B.5 Code for Measuring the Plausibility of a Scope Reading

```
public double computePlausibilityMeasure( Vector range1, Vector range2 )
      {
            double pl = 0.0;
            double summationPart = 0;
            int sz1 = range1.size();
            int sz2 = range2.size();
            int intersectSz = (intersect(range1,range2).size());
            int pr12 = preceed(range1,range2);
            int pr21 = preceed(range2,range1);

            if ( intersectSz > 0 )
            {
                  for ( int i=1; i<=intersectSz; i++ )
                  {
                        summationPart = summationPart + sz2 - pr21 - i + 1.0;
                  }
            }

            return (((pr12*sz2)+summationPart)*(1/(double)(sz1*sz2)));
      }
```

Appendix-C JAVA code of NLI-QSAR

```
import java.util.*;
import java.net.*;

// ***************************************************************************
// A Parser for a English sentences that perform
// syntactic/semantic analysis with a limited grammar...each
// syntactic object has a semantic object.
// ***************************************************************************


//--------------- Syntactic Objects

class LingObj
{
      // a class (global) attributes pointing to the dictionary/kb
      final static Dictionary    dictionary = new Dictionary();

      // every linguistic object has a meaning
      Meaning                    meaning             = new Meaning();

      // golbal attributes attributes
      boolean             parsedOK                   = false;
      Vector              restOfInput                = new Vector();
      Vector              input                      = new Vector();
      Vector              tokens                     = new Vector();

      String              parseTree                  = new String();
      int                 numOfNodes                 = 0;


      LingObj() {}
      LingObj(Vector words)
      {
            input.removeAllElements();
            input = (Vector)words.clone();
      }

      public void parse() { return; }

      //-------------------------------------------------------------------
      // This higher-order function corresponds to the BNF
      // functor '::='
      //-------------------------------------------------------------------

      static public LingObj consistsOf( Vector words, Vector interps )
      {
            LingObj       lObj                = new LingObj();//(words);
            boolean       failed              = false;
            Vector        foundTokens         = new Vector();
            int           compNumOfNodes      = 0;
            String        compParseTree       = new String("");
            Vector        text                = new Vector();
            text = (Vector)words.clone();

            for ( int i=0; ((i<interps.size()) && (!failed)); i++ )
            {
                  LingObj t = (LingObj)interps.elementAt(i);
```

```
                t.input.removeAllElements();
                t.restOfInput.removeAllElements();
                t.input = (Vector)text.clone();
                t.parse();

                if ( t.parsedOK )
                {
                        foundTokens = append(foundTokens, t.tokens);
                        compNumOfNodes = compNumOfNodes + t.numOfNodes;
                        compParseTree = compParseTree + t.parseTree;
                        text.removeAllElements();
                        text = (Vector)(t.restOfInput).clone();
                }
                else
                {
                        failed = true;
                        lObj.parsedOK = false;
                        lObj.restOfInput.removeAllElements();
                        lObj.restOfInput = words;//(Vector)text.clone();
                }
        }

        if ( !failed )
        {
                lObj.parsedOK = true;
                lObj.tokens = foundTokens;
                lObj.restOfInput.removeAllElements();
                lObj.restOfInput = (Vector)text.clone();
                lObj.parseTree = compParseTree;
        }

        return lObj;
}


//-------------------------------------------------------------------------
// this higher-order function corresponds to the BNF
// functor '|'
//-------------------------------------------------------------------------

public Vector orElse( Vector words, Vector cases )
{
        Vector result = new Vector();
        for ( int i=0; i<cases.size(); i++ )
        {
        Vector oneCase = (Vector)cases.elementAt(i);
        LingObj lObj = consistsOf( words, oneCase );
        result.addElement( lObj );
        }
        return result;
}

public Vector orElse( Vector words, Vector case1, Vector case2 )
{       Vector cases = new Vector();
        cases.addElement( case1 );
        cases.addElement( case2 );
        return orElse( words, cases );
}

public Vector orElse(Vector words,Vector case1,Vector case2,Vector case3)
{       Vector cases = new Vector();
        cases.addElement( case1 );
        cases.addElement( case2 );
```

```
        cases.addElement( case3 );
        return orElse( words, cases );
}

public Vector orElse( Vector words, Vector case1, Vector case2,
        Vector case3, Vector case4 )
{       Vector cases = new Vector();
        cases.addElement( case1 );
        cases.addElement( case2 );
        cases.addElement( case3 );
        cases.addElement( case4 );
        return orElse( words, cases );
}

public Vector rule( LingObj obj1 )
{       Vector result = new Vector();
        result.addElement( obj1 );
        return result;
}

public Vector rule( LingObj obj1, LingObj obj2 )
{       Vector result = new Vector();
        result.addElement( obj1 );
        result.addElement( obj2 );
        return result;
}

public Vector rule( LingObj obj1, LingObj obj2, LingObj obj3 )
{       Vector result = new Vector();
        result.addElement( obj1 );
        result.addElement( obj2 );
        result.addElement( obj3 );
        return result;
}

public Vector rule( LingObj obj1, LingObj obj2,LingObj obj3,LingObj obj4)
{       Vector result = new Vector();
        result.addElement( obj1 );
        result.addElement( obj2 );
        result.addElement( obj3 );
        result.addElement( obj4 );
        return result;
}

public Vector rule( LingObj obj1, LingObj obj2, LingObj obj3,
        LingObj obj4, LingObj obj5 )
{       Vector result = new Vector();
        result.addElement( obj1 );
        result.addElement( obj2 );
        result.addElement( obj3 );
        result.addElement( obj4 );
        result.addElement( obj5 );
        return result;
}

public Vector rule( LingObj obj1, LingObj obj2, LingObj obj3,
        LingObj obj4, LingObj obj5, LingObj obj6 )
{       Vector result = new Vector();
        result.addElement( obj1 );
        result.addElement( obj2 );
        result.addElement( obj3 );
        result.addElement( obj4 );
        result.addElement( obj5 );
```

```
            result.addElement( obj6 );
            return result;
        }

        public Vector rule( LingObj obj1, LingObj obj2, LingObj obj3,
            LingObj obj4, LingObj obj5, LingObj obj6, LingObj obj7 )
        {   Vector result = new Vector();
            result.addElement( obj1 );
            result.addElement( obj2 );
            result.addElement( obj3 );
            result.addElement( obj4 );
            result.addElement( obj5 );
            result.addElement( obj6 );
            result.addElement( obj7 );
            return result;
        }

        public void copyToThis( LingObj source, LingObj target )
        {
            target.restOfInput.removeAllElements();

            target.parsedOK             = source.parsedOK;
            target.tokens       = source.tokens;
            target.parseTree    = source.parseTree;
            target.restOfInput  = (Vector)(source.restOfInput).clone();
            target.numOfNodes   = source.numOfNodes;
        }

        static public Vector append(Vector vec1, Vector vec2)
        {   Vector result = new Vector();
            for ( int n=0; n<vec1.size(); n++ )
                result.addElement( vec1.elementAt(n) );
            for ( int m=0; m<vec2.size(); m++ )
                result.addElement( vec2.elementAt(m) );
            return result;
        }


        public void printWords( Vector words )
        {
            for ( int i=0; i<words.size(); i++ )
                System.out.print("["+(String)words.elementAt(i)+"] ");
        }

};


// ---------------------------------------------------------------------
// Terminal syntactic objects implemented as a class.
// ---------------------------------------------------------------------

class _ADJ extends LingObj
{
        Modifier      meaning = new Modifier();
        _ADJ( Vector words ) { super(words); }

        public void parse()
        {   if ( input.size() > 0 )
            {
                String token = (String)input.elementAt(0);
                meaning = dictionary.getMeaningOfAdj( token );
                if ( meaning != null )
                {
```

```
                        parsedOK = true;
                        parseTree = parseTree + "[ADJ  " + token + "]";
                        numOfNodes = 1;
                        tokens.addElement( token );
                        meaning.adjectives.removeAllElements();
                        meaning.adjectives.addElement( token );
                        for ( int i=1; i<input.size(); i++ )
                                restOfInput.addElement( input.elementAt(i) );
                    }
                    else
                    {
                        parsedOK = false;
                        restOfInput.removeAllElements();
                        restOfInput = (Vector)input.clone();
                    }
                }
        }
};

//-----------------------------------------------------------------------

class _NOUN extends LingObj
{
        Concept             meaning = new Concept();
        _NOUN( Vector words ) { super(words); }

        public void parse()
        {
                if ( input.size() > 0 )
                {
                        String word = (String)input.elementAt(0);
                        meaning = dictionary.getMeaningOfNoun( word );
                        if ( meaning != null )
                        {
                                parsedOK = true;
                                numOfNodes = 1;
                                parseTree = parseTree + "[NOUN  " + word + "]";
                                tokens.addElement( word );
                                for ( int i=1; i<input.size(); i++ )
                                        restOfInput.addElement( input.elementAt(i) );
                                meaning.nModifiers.removeAllElements();
                        }
                        else
                        {
                                parsedOK = false;
                                restOfInput = input;
                        }
                }
        }
};

//-----------------------------------------------------------------------

class _PREP extends LingObj
{
        _PREP( Vector words ) { super(words); }
        public void parse()
        {
                if ( input.size() > 0 )
                {
                        String token = (String)input.elementAt(0);
                        if ( (token.equals(new String("of")))   ||
                                (token.equals(new String("in"))) ||
```

```
                        (token.equals(new String("with")))||
                        (token.equals(new String("from")))||
                        (token.equals(new String("on"))) ||
                        (token.equals(new String("at")))
                    )
                {
                        parsedOK = true;
                        numOfNodes = 1;
                        parseTree = parseTree + "[PREP  " + token + "]";
                        tokens.addElement( token );
                        for ( int i=1; i<input.size(); i++ )
                                restOfInput.addElement( input.elementAt(i) );
                }
                else
                {
                        parsedOK = false;
                        restOfInput = input;
                }
        }
    }
};

//---------------------------------------------------------------------

class _DET extends LingObj
{
    Quantifier    meaning;
    String        quant         = new String("");
    int           flag          = 1;   //(0=numeric quantifier; 1=linguistic
                                       quantifier (the default))
    _DET( Vector words ) { super(words); }
    public void parse()
    {
        if ( input.size() > 0 )
        {
            String token = (String)input.elementAt(0);
            if ( isALinguisticQuantifier(token) ||
            isANumericQuantifier(token) )
            {
                parsedOK = true;
                numOfNodes = 1;
                parseTree = parseTree + "[DET  " + token + "]";
                tokens.addElement( token );
                for ( int i=1; i<input.size(); i++ )
                        restOfInput.addElement( input.elementAt(i) );
                meaning = new Quantifier( quant );
            }
            else
            {
                parsedOK = false;
                restOfInput = input;
            }
        }
    }

    private boolean isANumericQuantifier( String token )
    {
        try
        {
            Integer numValue = new Integer( token );
            quant=token;
            flag = 0;
            return true;
```

```
            }
            catch(NumberFormatException nfe)
            {
                  return false;
            }
      }

private boolean isALinguisticQuantifier( String token )
{
      if (token.equals(new String("no"))) { quant=token; return true; }
      if (token.equals(new String("any"))) { quant=token; return true; }
      if (token.equals(new String("every"))) { quant=token; return true; }
      if (token.equals(new String("a"))) { quant=token; return true; }
      if (token.equals(new String("an"))) { quant=token; return true; }
      if (token.equals(new String("the"))) { quant=token; return true; }
      if (token.equals(new String("few"))) { quant=token; return true; }
      if (token.equals(new String("several"))) { quant=token; return true; }
      if (token.equals(new String("some"))) { quant=token; return true; }

      if (token.equals(new String("many"))) { quant=token; return true; }
      if (token.equals(new String("most"))) { quant=token; return true; }
      if (token.equals(new String("all"))) { quant=token; return true; }
      if (token.equals(new String("one"))) { quant=token; return true; }
      if (token.equals(new String("two"))) { quant=token; return true; }
      if (token.equals(new String("three"))) { quant=token; return true; }
      if (token.equals(new String("four"))) { quant=token; return true; }
      if (token.equals(new String("five"))) { quant=token; return true; }
      if (token.equals(new String("six"))) { quant=token; return true; }
      if (token.equals(new String("seven"))) { quant=token; return true; }
      if (token.equals(new String("eight"))) { quant=token; return true; }
      if (token.equals(new String("nine"))) { quant=token; return true; }
      if (token.equals(new String("ten"))) { quant=token; return true; }
      return false;
      }

};

//---------------------------------------------------------------------

class _PNOUN extends LingObj
{
      Concept       meaning = new Concept();
      _PNOUN( Vector words ) { super(words); }

      public void parse()
      {
            if ( input.size() > 0 )
            {
                  String word = (String)input.elementAt(0);
                  meaning = dictionary.getMeaningOfPNoun( word );

                  if ( meaning != null )
                  {
                        parsedOK = true;
                        numOfNodes = 1;
                        parseTree = parseTree + "[PNOUN  " + word + "]";
                        tokens.addElement( word );
                        for ( int i=1; i<input.size(); i++ )
                              restOfInput.addElement( input.elementAt(i) );
                        meaning.constant = word;
                  }
                  else
                  {
```

```
                        parsedOK = false;
                        restOfInput = input;
                    }
                }
            }
};

//---------------------------------------------------------------------

class _TVERB extends LingObj
{
        Relation      meaning              = new Relation();
        Relation      negOnRelation        = new Relation();
        boolean       negation;
        _TVERB( Vector words ) { super(words); }

        public void parse()
        {
                NegationOnVerb();
                if ( input.size() > 0 )
                {
                        String token = (String)input.elementAt(0);
                        meaning = dictionary.getMeaningOfVerb( token );
                        if (negation == true)
                                meaning.negationOnRelation = true;
                        else
                                meaning.negationOnRelation = false;
                        if ( meaning != null )
                        {
                                parsedOK = true;
                                numOfNodes = 1;
                                parseTree = parseTree + "[VERBtr  " + token + "]";
                                tokens.addElement( token );
                                for ( int i=1; i<input.size(); i++ )
                                        restOfInput.addElement( input.elementAt(i) );
                        }
                        else
                        {
                                parsedOK = false;
                                restOfInput = input;
                        }
                }
        }

        public void NegationOnVerb()
        {       Vector text = new Vector();

                if ( ((String)input.elementAt(0)).compareTo("didnot") == 0 )
                        negation = true;
                else
                        negation = false;
                if (negation == true)
                {       System.out.print("\nNegation is on Relation1*\n");
                        for ( int i=1; i<input.size(); i++ )
                                text.addElement( input.elementAt(i) );
                        input = (Vector)text.clone();
                        text.removeAllElements();
                }

        }

};
```

```
//---------------------------------------------------------------------

class _RPNOUN extends LingObj
{
       _RPNOUN( Vector words ){ super(words); restOfInput.removeAllElements();}

       public void parse()
       {
              if ( input.size() > 0 )
              {      String token = (String)input.elementAt(0);
                     if ( (token.equals(new String("which")))       ||
                            (token.equals(new String("that")))      ||
                            (token.equals(new String("who")))       ||
                            (token.equals(new String("whose")))
                        )
                     {
                            parsedOK = true;
                            parseTree = parseTree + "[RELPRON  " + token + "]";
                            numOfNodes = 1;
                            tokens.addElement( token );
                            for ( int i=1; i<input.size(); i++ )
                                   restOfInput.addElement( input.elementAt(i) );
                     }
                     else
                     {
                            parsedOK = false;
                            restOfInput = input;
                     }
              }
       }
};


// ---------------------------------------------------------------------
// Non Terminal syntactic objects  implemented as classes.
// ---------------------------------------------------------------------

class _ADJS extends LingObj
{
       Modifier      meaning       = new Modifier();
       _ADJS( Vector words ) { super(words); tokens.removeAllElements(); }

       public void parse()
       {
              _ADJ adj;
              _ADJS adjs;

              adj = new _ADJ(input);
              adj.parse();

              if ( adj.parsedOK )
              {
                     parsedOK = true;
                     numOfNodes = 1;
                     String adjToken = (String)input.elementAt(0);
                     parseTree = parseTree + "[ADJS  " + adjToken + "]";
                     tokens.addElement( adjToken );
                     meaning = adj.meaning;
```

```
                    adjs = new _ADJS(adj.restOfInput);
                    adjs.parse();
                    if ( adjs.parsedOK )
                    {
                    Vector restOfAdjs = adjs.tokens;
                    tokens = append(tokens,restOfAdjs);
                    restOfInput = adjs.restOfInput;
                    for ( int i=0; i<restOfAdjs.size(); i++ )
                    parseTree=parseTree + " "+(String)restOfAdjs.elementAt (i);
                          meaning.addModifiers( adjs.tokens );
                    }
                    else
                    {
                    restOfInput = adj.restOfInput;
                    }
              }
              else
              {
                    parsedOK = false;
                    restOfInput = input;
                    tokens = new Vector();
              }
        }
};

//---------------------------------------------------------------------

class _NOUNS extends LingObj
{
      Concept             meaning      = new Concept();
      Vector              allConcepts  = new Vector();

      _NOUNS( Vector words )
      {
            super(words);
            allConcepts.removeAllElements();
      }

      public void parse()
      {
            _NOUN noun;
            _NOUNS nouns;

            noun = new _NOUN(input);
            noun.parse();

            if ( noun.parsedOK )
            {
                  parsedOK = true;
                  String nounToken = (String)input.elementAt(0);
                  Concept meaning1 = new Concept();
                  meaning1 = noun.meaning;
                  allConcepts.addElement( meaning1 );
                  parseTree = parseTree + "[NOUNS  " + nounToken + "]";
                  numOfNodes = 1;
                  tokens.addElement( nounToken );
                  nouns = new _NOUNS(noun.restOfInput);
                  nouns.parse();
                  if ( nouns.parsedOK )
                  {
                  Vector restOfNouns = nouns.tokens;
                  tokens = append(tokens,restOfNouns);
                  restOfInput = nouns.restOfInput;
```

```
                        for ( int i=0; i<restOfNouns.size(); i++ )
                        parseTree=parseTree+ " "+(String)restOfNouns.elementAt (i);
                        Vector firstConcept = new Vector();
                        firstConcept.addElement( meaning1 );
                        allConcepts = append( firstConcept, nouns.allConcepts );
                        meaning = nouns.meaning;
                        }
                        else
                        {
                              restOfInput = noun.restOfInput;
                              meaning = meaning1;
                        }

                        for ( int i=0; i<allConcepts.size()-1; i++ )
                        {(Concept)allConcepts.elementAt(i) );
                        meaning.addNModifier((Concept)allConcepts.element At(i));
                        }
                  }
                  else
                  {
                        parsedOK = false;
                        restOfInput = input;
                  }
            }
};

//---------------------------------------------------------------------

class _TP extends LingObj
{
      Concept     meaning     = new Concept();
      _NOUNS      nouns       = new _NOUNS(input);
      _ADJS       adjs        = new _ADJS(input);
      _PNOUN      pnoun       = new _PNOUN(input);

      _TP( Vector words ) { super(words); }

      public void parse()
      {
            Vector results = orElse( input
                              , rule( adjs, nouns )
                              , rule( nouns )
                              , rule( pnoun )
                              );

            LingObj lObj1 = (LingObj)results.elementAt(0);
            LingObj lObj2 = (LingObj)results.elementAt(1);
            LingObj lObj3 = (LingObj)results.elementAt(2);

      if ( lObj1.parsedOK ) { copyToThis( lObj1, this ); computeMeaning1(); }
      else if ( lObj2.parsedOK ){copyToThis( lObj2, this); computeMeaning2();}
      else if ( lObj3.parsedOK ){copyToThis(lObj3,this ); computeMeaning3();}
      else
      {
            parsedOK = false;
            restOfInput = input;
      }

      parseTree = "[TP  " + parseTree + "]";
      }

      // compositional semantic rules
      private void computeMeaning1()
```

```
        {       meaning = nouns.meaning;
                meaning.aModifiers = adjs.meaning;
        }

        private void computeMeaning2()
        {       meaning = nouns.meaning;
        }

        private void computeMeaning3()
        {       meaning = pnoun.meaning;
        }
};

//---------------------------------------------------------------------

class _NP0 extends LingObj
{
        QuantifiedConcept   meaning;
        _DET                det          = new _DET(input);
        _TP                 tp           = new _TP(input);
        _PNOUN              pnoun        = new _PNOUN(input);

        _NP0( Vector words ) { super(words); }

        public void parse()
        {
                Vector results = orElse( input
                                , rule( det, tp )
                                , rule( pnoun )
                                );

                LingObj lObj1 = (LingObj)results.elementAt(0);
                LingObj lObj2 = (LingObj)results.elementAt(1);

        if ( lObj1.parsedOK ) { copyToThis( lObj1, this ); computeMeaning1(); }
        else if ( lObj2.parsedOK ){copyToThis(lObj2, this); computeMeaning2();}
        else
        {
                parsedOK = false;
                restOfInput = input;
        }
        }

        // compositional semantic rules
        private void computeMeaning1()
        {       meaning = new QuantifiedConcept( det.meaning, tp.meaning );
                meaning.makeExpression( 0, false);
        }

        private void computeMeaning2()
        {       meaning = new QuantifiedConcept( pnoun.meaning );
                meaning.makeExpression( 0, false );
        }

};

//---------------------------------------------------------------------

class _NP extends LingObj
{
        QuantifiedConcept   meaning;
        _NP0                np0a   = new _NP0(input);
        _NP0                np0b   = new _NP0(input);
```

```
        _NP0                np0c   = new _NP0(input);
        _PP                 pp     = new _PP(input);
        _RCLAUSE            rclause= new _RCLAUSE(input);

        _NP( Vector words ) { super(words); }

        public void parse()
        {
              Vector results = orElse( input
                           , rule( np0a, pp )
                           , rule( np0b, rclause )
                           , rule( np0c )
                           );

              LingObj lObj1 = (LingObj)results.elementAt(0);
              LingObj lObj2 = (LingObj)results.elementAt(1);
              LingObj lObj3 = (LingObj)results.elementAt(2);

        if ( lObj1.parsedOK ) { copyToThis( lObj1, this ); computeMeaning1(); }
        else if(lObj2.parsedOK ){copyToThis(lObj2, this ); computeMeaning2(); }
        else if(lObj3.parsedOK){copyToThis( lObj3, this ); computeMeaning3();}
        else
        {
              parsedOK = false;
              restOfInput = input;
        }
              parseTree = "[NP  " + parseTree + "]";
        }

        // compositional semantic rules
        private void computeMeaning1()
        {     meaning = np0a.meaning;
              meaning.attach( pp.meaning );
        }

        private void computeMeaning3()
        {     meaning = np0c.meaning;
        }

};

//---------------------------------------------------------------------

class _PP extends LingObj
{
        Attachment    meaning;
        _PREP         prep          = new _PREP(input);
        _NP0          np            = new _NP0(input);

        _PP( Vector words ) { super(words); }

        public void parse()
        {
              LingObj lObj = consistsOf( input, rule( prep, np ) );
              parsedOK = lObj.parsedOK;
              restOfInput.removeAllElements();
              restOfInput = (Vector)(lObj.restOfInput).clone();
              tokens = lObj.tokens;
              parseTree = "[PP  " + lObj.parseTree + "]";
              numOfNodes = lObj.numOfNodes;
              computeMeaning();
        }
```

```
        // compositional semantic rules
        private void computeMeaning()
        {       if ( parsedOK )
                {
                        meaning = new Attachment(
                        (String)(prep.tokens).elementAt(0), np.meaning );
                }
        }

};

//---------------------------------------------------------------------

class _VP extends LingObj
{
        Clause          meaning;
        _TVERB          tverb  = new _TVERB(input);
        _NP                     np              = new _NP(input);

        _VP( Vector words ) { super(words); }

        public void parse()
        {
                LingObj lObj = consistsOf( input, rule( tverb, np ) );

                parsedOK = lObj.parsedOK;
                restOfInput.removeAllElements();
                restOfInput = (Vector)(lObj.restOfInput).clone();
                tokens = lObj.tokens;
                parseTree = "[VP  " + lObj.parseTree + "]";
                numOfNodes = lObj.numOfNodes;
                computeMeaning();
        }

        // compositional semantic rules
        private void computeMeaning()
        {
                if ( parsedOK )
                {
                        meaning = new Clause( tverb.meaning, np.meaning );
                }
                meaning.resolveAttachments();
        }
};

//---------------------------------------------------------------------

class _RCLAUSE extends LingObj
{
        _RCLAUSE( Vector words ) { super(words); }

        public void parse()
        {
                _RPNOUN             rpnoun = new _RPNOUN(input);
                _VP                 vp             = new _VP(input);

                LingObj lObj = consistsOf( input, rule( rpnoun, vp ) );

                parsedOK = lObj.parsedOK;
                restOfInput.removeAllElements();
                restOfInput = lObj.restOfInput;
                tokens = lObj.tokens;
                parseTree = "[RCLAUSE  " + lObj.parseTree + "]";
```

```
            numOfNodes = lObj.numOfNodes;
        }
};

//-------------------------------------------------------------------------

class _SENTENCE extends LingObj
{
        LogicalForm   meaning;
        _NP                 np          = new _NP(input);
        _VP                 vp          = new _VP(input);

        _SENTENCE( Vector words ) { super(words); }

        public void parse()
        {
                LingObj lObj = consistsOf( input, rule( np, vp ) );
                parsedOK = lObj.parsedOK;
                restOfInput.removeAllElements();
                restOfInput = (Vector)(lObj.restOfInput).clone();
                tokens = lObj.tokens;
                parseTree =  "[SENT  " + lObj.parseTree + "]";
                numOfNodes = lObj.numOfNodes;

                computeMeaning();
        }

        // compositional semantic rules
        private void computeMeaning()
        {
                if ( parsedOK )
                {
                        meaning = new LogicalForm( np.meaning, vp.meaning );
                }
        }

};
```

```
//*************************************************************************
// Semantic Objects of Corresponding Syntactic Objects
//*************************************************************************


import java.util.*;
import java.io.*;
import java.awt.*;

//-------------- Semantic Objects

class Meaning {};

class Modifier extends Meaning
{
       Vector adjectives = new Vector();
       Modifier() {}
       Modifier( String m )
       {
              adjectives.addElement( m );
       }

       public void addModifier( String adj ) { adjectives.addElement( adj ); }
       public void addModifiers( Modifier m )
       {
              Vector adjs = m.getAllModifiers();
              for ( int i=0; i< adjs.size(); i++ )
                     adjectives.addElement( (String)adjs.elementAt(i) );
       }

       public void addModifiers( /*String adj,*/ Vector adjs )
       {
              for ( int i=0; i< adjs.size(); i++ )
                     adjectives.addElement( (String)adjs.elementAt(i) );
       }

       public int numOfModifiers() { return adjectives.size(); }
       public Vector getAllModifiers() { return adjectives; }

};

//-------------------------------------------------------------------------

class Quantifier extends Meaning
{
       String quantifier    = new String("");
       String lingPart      = new String("");
       String numPart              = new String("");
       Vector range         = new Vector();

       Quantifier() {}
       Quantifier( String det )
       {
              quantifier = det;
       }
};

//-------------------------------------------------------------------------

class Expression {};

class Expression2 extends Expression
```

```
{
        String constant = new String("");
        Expression2( Concept pnoun )
        {
                constant = pnoun.constant;
        }

        public String getMatrix() { return constant; }
};

//---------------------------------------------------------------------------

class Expression1 extends Expression
{
        int             varCount        = 0;
        int             nextPredicate           = 1;
        String          quantifier      = new String("");
        String          variable        = new String("");
        String          predicate1      = new String("");
        String          predicate2      = new String("");
        String          connective      = new String("");
        String          matrix                  = new String("");

        Expression1( Quantifier quant )
        {
                quantifier = getQuantifier(quant.quantifier);
                variable = getVariable(varCount);
                connective = getConnective(quant.quantifier);
                predicate1 = (new String("P")) + (new
                             Integer(varCount+1)).toString();
                predicate2 = (new String("Q")) + (new
                             Integer(varCount+1)).toString();
        }

        Expression1( int vCount, Quantifier quant )
        {
                varCount = vCount;
                quantifier = getQuantifier(quant.quantifier);
                variable = getVariable(varCount);
                connective = getConnective(quant.quantifier);
                predicate1 = (new String("P")) + (new
                             Integer(vCount+1)).toString();
                predicate2 = (new String("Q")) + (new
                             Integer(vCount+1)).toString();
        }

        Expression1( int vCount, Quantifier quant, Concept concept, boolean
                                                                negationFlag )
        {
                varCount = vCount;
                variable = getVariable(varCount);
                quantifier = getQuantifier(quant.quantifier);
                connective = getConnective(quant.quantifier);
                predicate1 = getPredicate(concept);
                predicate2=((new String("Q"))+(new
                             Integer(vCount+1)).toString())+variable;
                if(negationFlag&( (quant.quantifier.toLowerCase()).compareTo("no")
                                  == 0 ))
                predicate2 = new String("not ").concat(predicate2);
                nextPredicate++;
        }

        Expression1( int vCount, Quantifier quant, Concept concept1, Concept
```

```
                                                                concept2 )
        {
                varCount = vCount;
                quantifier = getQuantifier(quant.quantifier);
                variable = getVariable(varCount);
                connective = getConnective(quant.quantifier);
                predicate1 = getPredicate(concept1);
                predicate2 = getPredicate(concept2);
                nextPredicate++;
        }

        private String getQuantifier(String quant)
        {
if ( (quant.toLowerCase()).compareTo("no") == 0 ) return (new String("FORALL"));
if ( (quant.toLowerCase()).compareTo("any") == 0 )return (new String("FORALL"));
if ( (quant.toLowerCase()).compareTo("every")== 0)return (new String("FORALL"));
if ( (quant.toLowerCase()).compareTo("each")== 0)return (new String("FORALL"));
if ( (quant.toLowerCase()).compareTo("all") == 0)return (new String("FORALL"));
if ( (quant.toLowerCase()).compareTo("some") == 0)return (new String("EXISTS"));
if ( (quant.toLowerCase()).compareTo("a") == 0)return (new String("EXISTS"));
if ( (quant.toLowerCase()).compareTo("an") == 0 ) return (new String("EXISTS"));
if ( (quant.toLowerCase()).compareTo("1") == 0 ) return (new String("EXISTS"));
if ( (quant.toLowerCase()).compareTo("2") == 0 ) return (new String("E!(2:)"));
if ( (quant.toLowerCase()).compareTo("3") == 0 ) return (new String("E!(3:)"));
if ( (quant.toLowerCase()).compareTo("4") == 0 ) return (new String("E!(4:)"));
if ( (quant.toLowerCase()).compareTo("5") == 0 ) return (new String("E!(5:)"));
if ( (quant.toLowerCase()).compareTo("6") == 0 ) return (new String("E!(6:)"));
if ( (quant.toLowerCase()).compareTo("few") == 0)return (new String("E!(few)"));
if ( (quant.toLowerCase()).compareTo("several")==0) return (newString
                                                        ("E!(several)"));
if ( (quant.toLowerCase()).compareTo("many") == 0 ) return (new
                                                        String("E!(many)"));
if ( (quant.toLowerCase()).compareTo("most") == 0 ) return (new
                                                        String("E!(most)"));
        return (new String("EXISTS"));
        }

        private String getVariable(int vCount)
        {
                String var = new String("(x");
                var = var + (new Integer(vCount+1)).toString() + ")";
                return var;
        }

        private String getConnective(String quant)
        {
if ( quant.compareTo("no") == 0 ) return (new String("->"));
if ( quant.compareTo("any") == 0 ) return (new String("->"));
if ( quant.compareTo("every") == 0 ) return (new String("->"));
if ( quant.compareTo("each") == 0 ) return (new String("->"));
if ( quant.compareTo("all") == 0 ) return (new String("->"));
if ( quant.compareTo("some") == 0 ) return (new String(" & "));
if ( quant.compareTo("a") == 0 ) return (new String(" & "));
if ( quant.compareTo("an") == 0 ) return (new String(" & "));
if ( quant.compareTo("1") == 0 ) return (new String(" & "));
if ( quant.compareTo("2") == 0 ) return (new String(" & "));
if ( quant.compareTo("3") == 0 ) return (new String(" & "));
if ( quant.compareTo("4") == 0 ) return (new String(" & "));
if ( quant.compareTo("5") == 0 ) return (new String(" & "));
if ( quant.compareTo("6") == 0 ) return (new String(" & "));
if ( quant.compareTo("few") == 0 ) return (new String(" & "));
if ( quant.compareTo("several") == 0 ) return (new String(" & "));
if ( quant.compareTo("many") == 0 ) return (new String(" & "));
```

```
if ( quant.compareTo("most") == 0 ) return (new String(" & "));
      return (new String(" & "));
      }

      private String getPredicate( Concept p )
      {
            String className = new String("");
            String predSymb = new String("");
            className = (p.getClass()).toString();
            predSymb = (className.substring(6)).toUpperCase();
            return predSymb + variable;
      }

      public void applyExpression( Expression1 expr, Boolean
                                                      negationInNounPhrase )
      {
        if (negationInNounPhrase)
        {
           String negationInExpr1 = new String(predicate2.substring(0,3));
           String negationInExpr2 = new String(expr.predicate2.substring(0,3));

           if ( negationInExpr1.compareTo("not") == 0 )
           {
              if ( (expr.quantifier.toUpperCase()).compareTo("FORALL") == 0 )
              {
                   expr.quantifier = new String("EXISTS");
                   expr.connective = "&";
               }
               else if((expr.quantifier.toUpperCase()).compareTo("EXISTS")==0 )
              {
                   expr.quantifier = new String("FORALL");
                   expr.connective = "->";
               }
               else if ( negationInExpr2.compareTo("not") == 0 )
                {

                }
           }
        }
        if ( nextPredicate == 1 ) predicate1 = expr.getMatrix();
            else predicate2 = expr.getMatrix();
      }


      public void unifyPrepRelation( String prep, Expression1 expr )
      {
            Expression1 nExpr = expr;
            nExpr.addPredicateConj( 2, prep, variable );
            predicate2 = nExpr.getMatrix();
      }

      private void addPredicateConj( int pos, String pred, String outsideVar )
      {
            if ( pos == 1 ) predicate1 = "(" + predicate1 + " & "
            + pred.toUpperCase() + "("+ outsideVar + ","   + variable + ")";
            else predicate2 = "(" + predicate2 + " & "
            + pred.toUpperCase() + "("+ outsideVar + ","   + variable + ")";
      }

      public void replaceBinaryRelation( Expression1 expr, String rel, boolean
      negationOnRelation, boolean negationInNounPhrase )
      {
            predicate2 = new String();
```

```
            predicate2 = rel + variable;
            if (negationOnRelation || negationInNounPhrase)
            predicate2 = new String("not ").concat(predicate2);

            String oldPred2 = new String("");
            oldPred2 = predicate2;
            int len = oldPred2.length();
            oldPred2 = oldPred2.substring(0,len-3)
                        + expr.variable.substring(1,3) + ","
                        + oldPred2.substring(len-3);
            predicate2 = oldPred2;
      }


      public void addAConceptConjunctToPredicate( Concept c )
      {     // by default we always add conjuncts to predicate 1
            String conj = getPredicate( c );
            int conjSymbLength = conj.length() - 4;
            conj = conj.substring( 0, conjSymbLength );
            predicate1 = conj + "-" + predicate1;
      }

      public void addAConceptDisjunctToPredicate( Concept c )
      {     // by default we always add disjuncts to predicate 1
      }


      public void addAnAdjConjunctToPredicate( String adj )
      {     // by default we always add conjuncts to predicate 1
            predicate1 = predicate1 + " & " + adj.toUpperCase() + variable;
      }


      public String getMatrix()
      {     matrix =   quantifier + variable +
                              "[" + predicate1
                                  + connective
                                  + predicate2 +
                               "]";

            return matrix;
      }

};

//---------------------------------------------------------------------------

class Concept extends Meaning
{
      String      constant            = new String("");
      char        number              = 's';
      double      typicalCardinality  = 1000000.0;
      Vector      nModifiers          = new Vector();     // noun modifiers
      Modifier    aModifiers          = new Modifier();   // adj modifiers

      Concept() {}
      Concept( double tc )
      {
            typicalCardinality = tc;
      }

      public void addNModifier( Concept concept)
      {
```

```
            nModifiers.addElement( concept );
        }

        public void updateTypicalCardinality( double tc )
        {
            typicalCardinality = tc;
        }

        public String getConceptName()
        {
            String result = new String();
            result = (this.getClass()).toString();
            return result.substring(5);
        }
};


class Person extends Concept { Person(double tc) {super(tc);} };
class Man extends Person { Man(double tc) {super(tc);} };
class Woman extends Person { Woman(double tc) {super(tc);} };
class Author extends Person { Author(double tc) {super(tc);} };

class Student extends Person
{
    Student(double tc) {super(tc);}
};

class Professor extends Person { Professor(double tc) {super(tc);} };
class Engineer extends Person { Engineer(double tc) {super(tc);} };
class Scientist extends Person { Scientist(double tc) {super(tc);} };
class Senator extends Person { Senator(double tc) {super(tc);} };
class President extends Person { President(double tc) {super(tc);} };
class Reviewer extends Person { Reviewer(double tc) {super(tc);} };
class Manager extends Person { Manager(double tc) {super(tc);} };
class Editor extends Person { Editor(double tc) {super(tc);} };
class Publisher extends Concept { Publisher(double tc) {super(tc);} };
class Subject extends Concept { Subject(double tc) {super(tc);} };
class Review extends Concept { Review(double tc) {super(tc);} };
class Date extends Concept { Date(double tc) {super(tc);} };
class Artifact extends Concept { Artifact(double tc) {super(tc);} };
class House extends Artifact { House(double tc) {super(tc);} };
class Shelf extends Artifact { Shelf(double tc) {super(tc);} };
class Piano extends Artifact { Piano(double tc) {super(tc);} };
class Computer extends Artifact { Computer(double tc) {super(tc);} };
class Chapter extends Concept { Chapter(double tc) {super(tc);} };
class Publication extends Artifact { Publication(double tc) {super(tc);} };
class Paper extends Publication { Paper(double tc) {super(tc);} };
class Article extends Publication { Article(double tc) {super(tc);} };
class Book extends Publication { Book(double tc) {super(tc);} };
class Journal extends Publication { Journal(double tc) {super(tc);} };
class Magazine extends Publication { Magazine(double tc) {super(tc);} };
class Newspaper extends Publication { Newspaper(double tc) {super(tc);} };
class QEvent extends Concept { QEvent(double tc) {super(tc);} };
class Conference extends QEvent { Conference(double tc) {super(tc);} };
class AAAIConference extends Conference{AAAIConference(double tc){super(tc);} };
class AAAI97 extends AAAIConference { AAAI97(double tc) {super(tc);} };
class ACLConference extends Conference { ACLConference(double tc){super(tc);} };
class ACL95 extends ACLConference { ACL95(double tc) {super(tc);} };
class Seminar extends QEvent { Seminar(double tc) {super(tc);} };
class Location extends Concept { Location(double tc) {super(tc);} };
class Street extends Location { Street(double tc) {super(tc);} };
class City extends Location { City(double tc) {super(tc);} };
class Floor extends Location { Floor(double tc) {super(tc);} };
```

```
class Country extends Location { Country(double tc) {super(tc);} };

//-------------------------------------------------------------------------

class Attachment extends Meaning
{
      String                    preposition = new String("");
      Relation                  relation;
      QuantifiedConcept         qConcept;

      Attachment( String prep, QuantifiedConcept qC )
      {
            preposition = prep;
            qConcept = qC;
      }
};

//-------------------------------------------------------------------------

class QuantifiedConcept extends Meaning
{
      Concept             concept;
      Expression1         expression1;
      Expression2         expression2;
      Quantifier          quantifier;
      Attachment          attachment;
      boolean                   ignoreAttachment = false;

      QuantifiedConcept( Concept con )
      {
            concept = con;
      }

      QuantifiedConcept( Quantifier quant, Concept con )
      {
            quantifier = quant;
            concept = con;
      }

      public boolean checkForNegation()
      {
            if ( (quantifier.quantifier.toLowerCase()).compareTo("no") == 0 )
                  return true;
            else
                  return false;
      }

      public Expression1 getExpression(int varCount, boolean
                                                      negationInNounPhrase)
      {
            Expression1 expr = new Expression1(varCount,quantifier,concept,
            negationInNounPhrase);
            makeExpression(varCount, negationInNounPhrase);
            expr.quantifier = expression1.quantifier;
            expr.variable = expression1.variable;
            expr.predicate1 = expression1.predicate1;
            expr.predicate2 = expression1.predicate2;
            expr.connective = expression1.connective;
            return expr;
      }

      public void makeExpression( int varCount, boolean negationInNounPhrase)
      {
```

```
        if ( quantifier != null )
        {
              Vector allAdjs = (concept.aModifiers).adjectives;
        expression1=new
        Expression1(varCount,quantifier,concept, negationInNounPhrase);

        // Update the predicates based on Noun Modifiers
        for ( int i=(concept.nModifiers).size()-1; i>=0; i-- )
        {
        Concept oneModConcept =
              (Concept)((concept.nModifiers).elementAt(i));
        expression1.addAConceptConjunctToPredicate( oneModConcept );
        }

        // Update the predicates based on Adjectives Modifiers
        for ( int i=allAdjs.size()-1; i>=0; i-- )
        {
              String oneAdj = (String)allAdjs.elementAt(i);
              expression1.addAnAdjConjunctToPredicate( oneAdj );
        }

        }
        else
        {
              expression2 = new Expression2( concept );
        }


}


public String getStrExpression()
{
      if ( quantifier != null )
            return expression1.getMatrix();
      else
            return expression2.getMatrix();
}

public void attach( Attachment a )
{
      attachment = a;
}

public String displayMeaning()
{
      String result = new String("");
      result = result + "[\n";
      result = result + "\tQUANTIFIER" + quantifier.quantifier
      + "\t[ NOUN" + (concept.getClass()).toString() + "\t  #MODIFIERS "
      + (new Integer(concept.nModifiers.size())).toString()
      + attachment.qConcept.displayMeaning() + "\t]\n" + "]\n";
      return result;
}

public double getModifiersEffect()
{
      double modifiedSize = (double)concept.typicalCardinality;

      // Each adjective reduces the typical cardinality by 10%
      Vector allAdjs = (concept.aModifiers).adjectives;
      for ( int i=allAdjs.size()-1; i>=0; i-- )
      {
```

```
                modifiedSize = modifiedSize - (0.1*modifiedSize);
            }

            // Each noun modifier reduces the typical cardinality by 15%
            for ( int i=(concept.nModifiers).size()-1; i>=0; i-- )
            {
                modifiedSize = modifiedSize - (0.15*modifiedSize);
            }

            return modifiedSize;
        }

};


//-------------------------------------------------------
// Quantified Concept; QC(R, C1, C2) = <m1, m2>
//-------------------------------------------------------

class QC  extends Meaning
{
        int         modality1    = 0;   // (0=possible; 1=typical; 2=necessary)
        int         modality2    = 0;   // (0=possible; 1=typical; 2=necessary)
        Vector      pair         = new Vector();
        Vector      range        = new Vector();

        QC() {}

        QC(String m1, String m2, int mod1, int mod2)
        {
            pair.addElement( m1 );
            pair.addElement( m2 );
            pair.addElement( new Integer(mod1) );
            pair.addElement( new Integer(mod2) );
        }

        public String getFirst() { return (String)pair.elementAt(0); }
        public String getSecond() { return (String)pair.elementAt(1); }
        public int getFirstMod(){return((Integer)pair.elementAt(2)).intValue();}
        public int getSecondMod(){return((Integer)pair.elementAt(3)).intValue();}
        public String displayQC(){return ("<"+getFirst()+","+getSecond()+">"); }
};


//-------------------------------------------------------
// Relation between two concepts; R(C1, C2)
//-------------------------------------------------------

class Relation extends Meaning
{
        int         tense        = 0;//(0=neutral; 1=past; 2=present; 3=future)
        int         form         = 0;// (0=active; 1=passive)
        String      relation     = new String();
        Vector      qcs          = new Vector();
        QC          defaultQC;//  = new QC("1+","1+",1,1);
        boolean     negationOnRelation;

        Relation() {}
        Relation( int t )
        {
            tense = t;
        }
```

```
        Relation( int t, int f )
        {
                tense = t;
                form = f;
        }

        public boolean checkNegationOnRelation()
        {
                if(negationOnRelation)
                        return true;
                else
                        return false;
        }

        void setTense( int t ) { tense = t; }
        void setForm( int f ) { form = f; }

        public void makeQC( Concept concept1, Concept concept2, QC qc )
        {
                Vector pair = new Vector();
                pair.addElement(concept1.getConceptName());
                pair.addElement(concept2.getConceptName());
                Vector oneQC = new Vector();
                oneQC.addElement( pair );
                oneQC.addElement( qc );
        }

        public QC getQC(Concept concept1, Concept concept2)
        {
                return defaultQC;
        }

//-------------------------------------------------------------
// Determine if an object is of a certain type USING INHERITENCE
// i.e., isOfType(obj,TYPE) is true if TYPE(obj)=TYPE or if also
// if obj is child (subtype) of any object of type TYPE!
//-------------------------------------------------------------

        public boolean isOfType( Object obj, String name )
        {
                String objType = obj.getClass().getName();
                if ( objType.compareTo(name) == 0 ) return true;
                Vector allSuperClasses = getAllSuperClasses( obj );
                return allSuperClasses.contains( name );
        }

        public Vector getAllSuperClasses( Object obj )
        {
                Vector result = new Vector();
                Object objNext = new Object();

                objNext = obj;
                Class classNext = objNext.getClass();
                String objNextName = new String();
                objNextName = objNext.getClass().getSuperclass().getName();

                while ( objNextName.compareTo("Concept") != 0 )
                {
                        result.addElement( objNextName );
                        classNext = classNext.getSuperclass();

                        try
                        {
```

```
                        objNextName = classNext.getName();
                    }
                    catch(Exception ie)
                    {
                        System.out.print("\n[Relation::getAllSuperClasses]:
                            "+    "Instance could not be created.");
                        return result;
                    }
                }
                result.addElement(new String("Concept"));
                return result;
        }
};

//-------------------------------------------------------------------------

class Lift extends Relation
{
        Lift( int t )
        {
                super(t);
                if ( t == 1 ) relation = "Lifted";
                else relation = "Lift";

                super.defaultQC = new QC("1+","1+",1,1);
        }

        public QC getQC(Concept concept1, Concept concept2)
        {

                //*********************************************************
                // QC(Lift,Person,Piano)=<2+,1
                //*********************************************************
                if ( isOfType(concept1,"Person") && isOfType(concept2,"Piano") )
                        return new QC("2+","1",1,1);

                //*********************************************************
                // QC(Lift,Person,Publication)=<1,1+>
                //*********************************************************
        if ( isOfType(concept1,"Person") && isOfType(concept2,"Publication") )
                return new QC("1","1+",1,1);

                return super.defaultQC;
        }
};

//-------------------------------------------------------------------------

class Write extends Relation
{
        Write( int t )
        {
                super(t);
                if ( t == 1 ) relation = "Wrote";
                else relation = "Write";

                super.defaultQC = new QC("1+","1+",1,1);
        }

        public QC getQC(Concept concept1, Concept concept2)
        {
                //*********************************************************
                // QC(Write,Person,Article)=<1,1+>
```

```
            //*********************************************************************
            if ( isOfType(concept1,"Person") && isOfType(concept2,"Article") )
                  return new QC("1","1+",1,1);

            //*********************************************************************
            // QC(Lift,Person,Publication)=<1,1+>
            //*********************************************************************
      if ( isOfType(concept1,"Person") && isOfType(concept2,"Publication") )
                  return new QC("1","1+",1,1);

            return super.defaultQC;
      }
};

//-----------------------------------------------------------------------------

class Submit extends Relation
{
      Submit( int t )
      {
            super(t);
            if ( t == 1 ) relation = "Submitted";
            else relation = "Submit";

            super.defaultQC = new QC("1+","1+",1,1);
      }

      public QC getQC(Concept concept1, Concept concept2)
      {
            //*********************************************************************
            // QC(Lift,Person,Piano)=<2+,1>
            //*********************************************************************
            if ( isOfType(concept1,"Person") && isOfType(concept2,"Piano") )
                  return new QC("2+","1+",1,1);

            //*********************************************************************
            // QC(Lift,Person,Publication)=<few,1+>
            //*********************************************************************
      if ( isOfType(concept1,"Person") && isOfType(concept2,"Publication") )
                  return new QC("few","1+",1,1);

            return super.defaultQC;
      }
};

//-----------------------------------------------------------------------------

class Attend extends Relation
{
      Attend( int t )
      {
            super(t);
            if ( t == 1 ) relation = "Attended";
            else relation = "Attend";

            super.defaultQC = new QC("1+","1+",1,1);
      }


      public QC getQC(Concept concept1, Concept concept2)
      {
            //*********************************************************************
```

```
            // QC(Lift,Person,Seminar)=<1+,1+>

            //*********************************************************************
            if ( isOfType(concept1,"Person") && isOfType(concept2,"Seminar") )
                  return new QC("1+","1+",1,1);

            return super.defaultQC;
      }
};

//-----------------------------------------------------------------------------

class Give extends Relation
{
      Give( int t )
      {
            super(t);
            if ( t == 1 ) relation = "Gave";
            else relation = "Give";

            super.defaultQC = new QC("1","1+",1,1);
      }

      public QC getQC(Concept concept1, Concept concept2)
      {
            //*********************************************************************
            // QC(Give,Person,Seminar)=<1,1+>
            //*********************************************************************
            if ( isOfType(concept1,"Person") && isOfType(concept2,"Seminar") )
                  return new QC("1","1+",1,1);

            //*********************************************************************
            // QC(Lift,Person,Publication)=<1,1+>
            //*********************************************************************
      if ( isOfType(concept1,"Person") && isOfType(concept2,"Publication") )
                  return new QC("1","1+",1,1);

            return super.defaultQC;
      }
};

//-----------------------------------------------------------------------------

class Clause extends Meaning
{
      Relation            relation;
      QuantifiedConcept   qConcept;
      Vector              attachments;

      Clause( Relation r, QuantifiedConcept qc )
      {
            relation = r;
            qConcept = qc;
      }

      public void resolveAttachments()
      {
            boolean notDone = true;
            while ( !notDone )
            {
                  if ( qConcept.attachment != null )
                  {
                  }
```

```
            }
        }
};


//*********************************************************
// Computing the meaning of a sentence
//*********************************************************

class LogicalForm extends Meaning
{
        Relation            relation;
        QuantifiedConcept   qConcept1;
        QuantifiedConcept   qConcept2;
        Vector              readings = new Vector();
        boolean             negationInNounPhrase = false;


        LogicalForm( QuantifiedConcept qc, Clause c )
        {
                relation = c.relation;
                qConcept1 = qc;
                qConcept2 = c.qConcept;
        }


        public void checkNegation()
        {
                negationInNounPhrase = qConcept1.checkForNegation() ||
                                       qConcept2.checkForNegation();
        }

        public String getLogicalForm( QuantifiedConcept qConcept1,
                                      QuantifiedConcept qConcept2 )
        {
                Expression1 expr1a = qConcept1.getExpression(0,
                                                     negationInNounPhrase);
                Expression1 expr2a = qConcept2.getExpression(1,
                                                     negationInNounPhrase);
                expr2a.replaceBinaryRelation( expr1a, relation.relation,
                        relation.checkNegationOnRelation(), negationInNounPhrase);
                expr1a.applyExpression( expr2a, negationInNounPhrase);
                return expr1a.getMatrix();
        }

        public String getScopeNeutralLF()
        {
                String finalRes = new String();
                finalRes = finalRes + "[\n";
                finalRes = finalRes + "\t"
                                    + getLogicalForm(qConcept1,qConcept2)+";\n\t"
                                    + getLogicalForm(qConcept2,qConcept1);
                finalRes = finalRes + "\n]\n";
                return finalRes;
        }

        public String getScopedLF()
        {
                String result = new String("");
                return result;
        }

};
```

```
//*************************************************************
// The dictionary
//*************************************************************


class Dictionary
{

      static Hashtable    _nouns = new Hashtable();
      static Hashtable    _pnouns = new Hashtable();
      static Hashtable    _adjs = new Hashtable();
      static Hashtable    _tverbs = new Hashtable();

      Person        person        = new Person(100.0);
      Man           man           = new Man(100.0);
      Woman         woman         = new Woman(100.0);
      Book          book          = new Book(100.0);
      Author        author        = new Author(100.0);
      Subject       subject       = new Subject(100.0);
      Review        review        = new Review(100.0);
      Date          date          = new Date(100.0);
      Reviewer      reviewer      = new Reviewer(100.0);
      Editor        editor        = new Editor(100.0);
      Publisher     publisher     = new Publisher(100.0);
      Student       student       = new Student(100.0);
      Professor     professor     = new Professor(100.0);
      Engineer      engineer      = new Engineer(100.0);
      President     president     = new President(100.0);
      Manager       manager       = new Manager(100.0);
      Artifact      artifact      = new Artifact(100.0);
      House         house         = new House(100.0);
      Piano         piano         = new Piano(100.0);
      Computer      computer      = new Computer(100.0);
      Chapter       chapter       = new Chapter(100.0);
      Paper         paper         = new Paper(100.0);
      Article       article       = new Article(100.0);
      Publication   publication   = new Publication(100.0);
      Journal       journal       = new Journal(100.0);
      Magazine      magazine      = new Magazine(100.0);
      Newspaper     newspaper     = new Newspaper(100.0);
      Conference    conference    = new Conference(100.0);
      Seminar       seminar       = new Seminar(100.0);
      Street        street        = new Street(100.0);
      City          city          = new City(100.0);
      Location      location      = new Location(100.0);
      Scientist     scientist     = new Scientist(100.0);
      Senator       senator       = new Senator(100.0);
      Country       country       = new Country(100.0);
      Floor         floor         = new Floor(100.0);
      Shelf         shelf         = new Shelf(100.0);
      QEvent        qEvent        = new QEvent(100.0);

      Dictionary()
      {
            makeEntries();
      }

      public void makeEntries()
      {
            // common _nouns
            _nouns.put(new String("people"), singular( person )   );
            _nouns.put(new String("person"), singular( person )   );
            _nouns.put(new String("persons"), plural( person )    );
```

```
_nouns.put(new String("man"),     singular( man ) );
_nouns.put(new String("men"),     plural( man ) );
_nouns.put(new String("woman"), singular( woman ) );
_nouns.put(new String("women"), plural( woman ) );
_nouns.put(new String("book"), singular( book ) );
_nouns.put(new String("books"), plural( book ) );
_nouns.put(new String("magazine"), singular( magazine ) );
_nouns.put(new String("magazines"), plural( magazine ) );
_nouns.put(new String("journal"), singular( journal ) );
_nouns.put(new String("journals"), plural( journal )  );
_nouns.put(new String("chapter"), singular( chapter ) );
_nouns.put(new String("chapters"), plural( chapter )  );
_nouns.put(new String("article"), singular( article ) );
_nouns.put(new String("articles"), plural( article )  );
_nouns.put(new String("paper"), singular( paper ) );
_nouns.put(new String("papers"), plural( paper ) );
_nouns.put(new String("newspaper"), singular( newspaper ) );
_nouns.put(new String("newspapers"),    plural( newspaper ) );
_nouns.put(new String("author"), singular( author ) );
_nouns.put(new String("authors"), plural( author ) );
_nouns.put(new String("subject"), singular( subject ) );
_nouns.put(new String("subjects"), plural( subject ) );
_nouns.put(new String("review"), singular( review ) );
_nouns.put(new String("reviews"), plural( review ) );
_nouns.put(new String("date"), singular(date) );
_nouns.put(new String("dates"), plural( date ) );
_nouns.put(new String("reviewer"), singular( reviewer ) );
_nouns.put(new String("reviewers"), plural( reviewer ) );
_nouns.put(new String("student"), singular( student ) );
_nouns.put(new String("students"), plural( student )  );
_nouns.put(new String("engineer"), singular( engineer ) );
_nouns.put(new String("engineers"), plural( engineer ) );
_nouns.put(new String("professor"), singular( professor ) );
_nouns.put(new String("professors"), plural( professor ) );
_nouns.put(new String("faculty"), singular( professor ) );
_nouns.put(new String("president"), plural( president ) );
_nouns.put(new String("presidents"), singular( president ) );
_nouns.put(new String("manager"), singular( manager ) );
_nouns.put(new String("managers"), plural( manager )  );
_nouns.put(new String("editor"),singular( editor ) );
_nouns.put(new String("editors"), plural( editor ) );
_nouns.put(new String("publisher"), singular( publisher ) );
_nouns.put(new String("publishers"), plural( publisher ) );
_nouns.put(new String("house"), singular( house ) );
_nouns.put(new String("houses"), plural( house ) );
_nouns.put(new String("street"), singular( street ) );
_nouns.put(new String("streets"), plural( street ) );
_nouns.put(new String("city"), singular( city ) );
_nouns.put(new String("cities"), plural( city ) );
_nouns.put(new String("piano"), singular( piano ) );
_nouns.put(new String("pianos"), plural( piano ) );
_nouns.put(new String("conference"), singular( conference ) );
_nouns.put(new String("conferences"), plural( conference ) );
_nouns.put(new String("seminar"), singular( seminar ) );
_nouns.put(new String("seminars"), plural( seminar ) );
_nouns.put(new String("location"), singular( location ) );
_nouns.put(new String("locations"), plural( location ) );
_nouns.put(new String("scientist"), singular( scientist ) );
_nouns.put(new String("scientists"), plural( scientist ) );
_nouns.put(new String("country"), singular( country ) );
_nouns.put(new String("countries"), plural( country ) );
_nouns.put(new String("senator"), singular( senator ) );
_nouns.put(new String("senstors"), plural( senator )  );
```

```
        _nouns.put(new String("shelf"), singular( shelf ) );
        _nouns.put(new String("shelves"), plural( shelf ) );
        _nouns.put(new String("floor"), singular( floor ) );
        _nouns.put(new String("floors"),plural( floor ) );
        _nouns.put(new String("computer"), singular( computer ) );
        _nouns.put(new String("computers"), plural( computer ) );

        // transitive verbs
        _tverbs.put(new String("write"), new Write(0) );
        _tverbs.put(new String("writes"), new Write(0) );
        _tverbs.put(new String("wrote"), new Write(1) );
        _tverbs.put(new String("written"), new Write(1) );
        _tverbs.put(new String("author"), new Write );
        _tverbs.put(new String("authored"), new Write(1) );
        _tverbs.put(new String("authors"), new Write(0) );
        _tverbs.put(new String("review"), new vReview(0) );
        _tverbs.put(new String("reviewes"), new vReview(0) );
        _tverbs.put(new String("reviewed"), new vReview(1) );
        _tverbs.put(new String("publish"), new Publish(0) );
        _tverbs.put(new String("publishes"), new Publish(0) );
        _tverbs.put(new String("published"), new Publish(1) );
        _tverbs.put(new String("lift"), new Lift(0) );
        _tverbs.put(new String("lifts"), new Lift(0) );
        _tverbs.put(new String("lifted"), new Lift(1) );
        _tverbs.put(new String("attend"), new Attend(0) );
        _tverbs.put(new String("attends"), new Attend(0) );
        _tverbs.put(new String("attended"), new Attend(1) );
        _tverbs.put(new String("submit"), new Submit(0) );
        _tverbs.put(new String("submits"), new Submit(0) );
        _tverbs.put(new String("submitted"), new Submit(1) );
        _tverbs.put(new String("give"), new Give(0) );
        _tverbs.put(new String("gives"), new Give(0) );
        _tverbs.put(new String("gave"), new Give(1) );
        _tverbs.put(new String("given"), new Give(1) );
        _tverbs.put(new String("advertise"), new Advertise(0) );
        _tverbs.put(new String("advertises"),    new Advertise(0) );
        _tverbs.put(new String("advertised"),    new Advertise(1) );
        _tverbs.put(new String("visit"), new Visit(0) );
        _tverbs.put(new String("visits"), new Visit(0) );
        _tverbs.put(new String("visited"), new Visit(1) );
        _tverbs.put(new String("consult"), new Consult(0) );
        _tverbs.put(new String("consults"), new Consult(0) );
        _tverbs.put(new String("consulted"), new Consult(1) );

        // adjectives
        _adjs.put(new String("famous"), new Modifier("famous") );
        _adjs.put(new String("senior"), new Modifier("senior") );
        _adjs.put(new String("associate"), new Modifier("associate") );
        _adjs.put(new String("assistant"), new Modifier("assistant") );
        _adjs.put(new String("old"), new Modifier("old") );
        _adjs.put(new String("graduate"), new Modifier("graduate") );

        // proper names
        _pnouns.put(new String("John"), instanceOf( person )  );
        _pnouns.put(new String("MIT"), instanceOf( location ) );
        _pnouns.put(new String("ACL-95"), instanceOf( conference ) );

}

public Concept singular( Concept e ) { e.number = 's'; return e; }
public Concept plural ( Concept e ) { e.number = 'p'; return e; }

public Concept instanceOf( Concept e )
```

```
        {
                try
                {
                        Concept newConcept = e;//(Concept)e.clone();
                        newConcept.number = 's';
                        newConcept.typicalCardinality = 1;
                        return newConcept;
                }
                catch(Exception cnse)
                {
                        return new Concept();
                }
        }

        public Concept getMeaningOfNoun( String word )
        {
                return (Concept)_nouns.get( word );
        }

        public Relation getMeaningOfVerb( String word )
        {
                return (Relation)_tverbs.get( word );
        }

        public Modifier getMeaningOfAdj( String word )
        {
                return (Modifier)_adjs.get( word );
        }

        public Concept getMeaningOfPNoun( String word )
        {
                return (Concept)_pnouns.get( word );
        }

};
```

```
import java.applet.*;
import java.awt.*;
import java.awt.Event;
import java.util.*;
import QCDemoFrame;
import DialogLayout;
import IDD_DIALOG1;
import Interpreter;
import Dlg1;

public class QCDemo extends Applet implements Runnable
{
      private Thread       m_QCDemo = null;
      IDD_DIALOG1          mainWindow;

// STANDALONE APPLICATION SUPPORT:
//m_fStandAlone will be set to true if applet is run standalone
//-------------------------------------------------------------
      private boolean m_fStandAlone = true;

      public static void main(String args[])
      {
            QCDemoFrame frame = new QCDemoFrame("QCDemo");

// Must show Frame before we size it so insets() will return valid values
//-------------------------------------------------------------
            frame.show();
            frame.hide();
            frame.resize(frame.insets().left + frame.insets().right  +
            850, frame.insets().top  + frame.insets().bottom + 430);
            QCDemo applet_QCDemo = new QCDemo();
            frame.add("Center", applet_QCDemo);
            applet_QCDemo.m_fStandAlone = true;
            applet_QCDemo.init();
            applet_QCDemo.start();
            frame.show();
      }

      public QCDemo()
      {
      }

      public void init()
      {
            resize(600, 600);
            mainWindow = new IDD_DIALOG1(this);
            mainWindow.CreateControls();
            mainWindow.RelationsinKB.addItem( new String("Lift") );
            mainWindow.RelationsinKB.addItem( new String("LocatedOn"));
            mainWindow.RelationsinKB.addItem( new String("Advertise"));
            mainWindow.RelationsinKB.addItem( new String("Submit") );
            mainWindow.RelationsinKB.addItem( new String("Attend") );
            mainWindow.RelationsinKB.addItem( new String("Publish") );
            mainWindow.RelationsinKB.addItem( new String("Submit") );
            mainWindow.RelationsinKB.addItem( new String("Review") );
            mainWindow.RelationsinKB.addItem( new String("Visit") );
            mainWindow.RelationsinKB.addItem( new String("Write") );
            mainWindow.RelationsinKB.addItem( new String("Author") );
            mainWindow.QCsForRelation.addItem( new
                              String("QC(House,Stree)=<many,1>") );
            mainWindow.QCsForRelation.addItem( new
                              String("QC(Book,Shelf)=<many,1>") );
            StringTokenizer strOfTokens;
```

```
            Vector words = new Vector();
            strOfTokens = new StringTokenizer("every man lifted a book"," .");
            int count = strOfTokens.countTokens();
            for ( int i=0; i<count; i++ )
            {
                  String word = strOfTokens.nextToken();
                  words.addElement( word );
            }

            _SENTENCE obj = new _SENTENCE( words );
            System.out.println("Parsing " + words);
            obj.parse();
            if ( obj.parsedOK )
            {
                  System.out.print("\nParse succeeded...");
                  System.out.print("\nRest of Input = " + obj.restOfInput);
                  System.out.print("\nScope Nuetral Logical Form:\n" +
                              obj.meaning.getScopeNeutralLF() );
            }
            else System.out.print("\nParse failed...");
      }


      public void start()
      {
      }

// This is a part of the Microsoft Visual J++ library.
// Copyright (C) 1996 Microsoft Corporation
// All rights reserved.

import java.util.Hashtable;
import java.awt.LayoutManager;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.FontMetrics;
import java.awt.Insets;
import java.awt.Label;

// class DialogLayout

// This class allows you to associate a Rectangle (x, y, width, height) with a
// Component in a Container.  If called upon to layout the container, this
// layout manager will layout based on the translation of dialog units to
// pixels.

public class DialogLayout
      implements LayoutManager
{
      protected Hashtable m_map = new Hashtable();
      protected int m_width;
      protected int m_height;

      // DialogLayout methods

      public DialogLayout(Container parent, int width, int height)
      {
            Construct(parent, width, height);
      }

      public DialogLayout(Container parent, Dimension d)
```

```
{
        Construct(parent, d.width, d.height);
}

public void setShape(Component comp, int x, int y, int width, int height)
{
        m_map.put(comp, new Rectangle(x, y, width, height));
}

public void setShape(Component comp, Rectangle rect)
{
m_map.put(comp, new Rectangle(rect.x, rect.y, rect.width, rect.height));
}

public Rectangle getShape(Component comp)
{
        Rectangle rect = (Rectangle)m_map.get(comp);
        return new Rectangle(rect.x, rect.y, rect.width, rect.height);
}

public Dimension getDialogSize()
{
        return new Dimension(m_width, m_height);
}

// LayoutManager Methods

public void addLayoutComponent(String name, Component comp) { }
public void removeLayoutComponent(Component comp) { }

public Dimension preferredLayoutSize(Container parent)
{
        return new Dimension(m_width, m_height);
}

public Dimension minimumLayoutSize(Container parent)
{
        return new Dimension(m_width, m_height);
}

public void layoutContainer(Container parent)
{
        int count = parent.countComponents();
        Rectangle rect = new Rectangle();
        int charHeight = getCharHeight(parent);
        int charWidth = getCharWidth(parent);
        Insets insets = parent.insets();
        FontMetrics m = parent.getFontMetrics(parent.getFont());

        for (int i = 0; i < count; i++)
        {
                Component c = parent.getComponent(i);
                Rectangle r = (Rectangle)m_map.get(c);
                if (r != null)
                {
                        rect.x = r.x;
                        rect.y = r.y;
                        rect.height = r.height;
                        rect.width = r.width;
                        mapRectangle(rect, charWidth, charHeight);
                        if (c instanceof Label)
                        {
                                // Adjusts for space at left of Java labels.
```

```
                        rect.x     -= 12;
                        rect.width += 12;
                    }

                    rect.x += insets.left;
                    rect.y += insets.top;
                    c.reshape(rect.x, rect.y, rect.width, rect.height);
                }
            }
        }

        // Implementation Helpers

        protected void Construct(Container parent, int width, int height)
        {
            Rectangle rect = new Rectangle(0, 0, width, height);
            mapRectangle(rect, getCharWidth(parent), getCharHeight(parent));
            m_width = rect.width;
            m_height = rect.height;
        }

        protected int getCharWidth(Container parent)
        {
            FontMetrics m = parent.getFontMetrics(parent.getFont());
            String s = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
            int    width = m.stringWidth(s) / s.length();

            if (width <= 0)
                width = 1;
            return width;
        }

        protected int getCharHeight(Container parent)
        {
            FontMetrics m = parent.getFontMetrics(parent.getFont());
            int height = m.getHeight();
            return height;
        }

        protected void mapRectangle(Rectangle rect, int charWidth, int
                                                                charHeight)
        {
            rect.x      = (rect.x      * charWidth)  / 4;
            rect.y      = (rect.y      * charHeight) / 8;
            rect.width  = (rect.width  * charWidth)  / 4;
            rect.height = (rect.height * charHeight) / 8;
        }
}
//---------------------------------------------------------------------------
// IDD_DIALOG1.java:
// Implementation for container control initialization class IDD_DIALOG1
// This class can be use to create controls within any container.
//---------------------------------------------------------------------------
import java.awt.*;
import DialogLayout;

public class IDD_DIALOG1
{
        Container    m_Parent       = null;
        boolean      m_fInitialized = false;
        DialogLayout m_Layout;

        // Control definitions
```

```
        //---------------------------------------------------------------
        TextArea        InputText;
        Button          ShowParse;
        Button          ShowRelevantQCs;
        Button          ProcessSentence;
        TextArea        OutputField;
        Button          ChangeQC;
        Button          Exit;
        Choice          RelationsinKB;
        TextField       IDC_EDIT3;
        List            QCsForRelation;
        Label           IDC_STATIC1;
        Label           IDC_STATIC2;
        Button          Clear;
        Button          ShowLogicalForm;


        // Constructor
        //---------------------------------------------------------------
        public IDD_DIALOG1 (Container parent)
        {
              m_Parent = parent;
        }

        // Initialization.
        //---------------------------------------------------------------
        public boolean CreateControls()
        {
              // Can only init controls once
              //-----------------------------------------------------------
              if (m_fInitialized || m_Parent == null)
                    return false;

              // Parent must be a derivation of the Container class
              //-----------------------------------------------------------
              if (!(m_Parent instanceof Container))
                    return false;

// Since there is no way to know if a given font is supported from platform to
// platform, we only change the size of the font, and not type face name.  And,
// we only change the font if the dialog resource specified a font.
//---------------------------------------------------------------
              Font OldFnt = m_Parent.getFont();

              if (OldFnt != null)
              {
                    Font NewFnt=new Font(OldFnt.getName(),OldFnt.getStyle(),8);
                    m_Parent.setFont(NewFnt);
              }

              // All position and sizes are in Dialog Units, so, we use the
              // DialogLayout manager.
              //-----------------------------------------------------------
              m_Layout = new DialogLayout(m_Parent, 562, 284);
              m_Parent.setLayout(m_Layout);
              m_Parent.addNotify();

              Dimension size   = m_Layout.getDialogSize();
              Insets    insets = m_Parent.insets();

              m_Parent.resize(insets.left + size.width  + insets.right,
                         insets.top  + size.height + insets.bottom);
```

```
        // Control creation
        //--------------------------------------------------------------------
        InputText = new TextArea ("");
        m_Parent.add(InputText);
        m_Layout.setShape(InputText, 99, 33, 236, 57);

        ShowParse = new Button ("Parse Tree");
        m_Parent.add(ShowParse);
        m_Layout.setShape(ShowParse, 13, 34, 83, 14);

        ShowRelevantQCs = new Button ("Relevant QCs");
        m_Parent.add(ShowRelevantQCs);
        m_Layout.setShape(ShowRelevantQCs, 13, 48, 83, 14);

        ProcessSentence = new Button ("Scope");
        m_Parent.add(ProcessSentence);
        m_Layout.setShape(ProcessSentence, 13, 76, 83, 13);

        OutputField = new TextArea ("");
        m_Parent.add(OutputField);
        m_Layout.setShape(OutputField, 99, 106, 236, 135);

        ChangeQC = new Button ("Change QC");
        m_Parent.add(ChangeQC);
        m_Layout.setShape(ChangeQC, 493, 122, 50, 12);

        Exit = new Button ("Exit");
        m_Parent.add(Exit);
        m_Layout.setShape(Exit, 466, 225, 80, 17);

        RelationsinKB = new Choice ();
        m_Parent.add(RelationsinKB);
        m_Layout.setShape(RelationsinKB, 360, 35, 69, 13);

        IDC_EDIT3 = new TextField ("");
        m_Parent.add(IDC_EDIT3);
        m_Layout.setShape(IDC_EDIT3, 432, 122, 59, 12);

        QCsForRelation = new List (1, false);
        m_Parent.add(QCsForRelation);
        m_Layout.setShape(QCsForRelation, 431, 35, 112, 84);

        IDC_STATIC1 = new Label ("Binary Relation", Label.LEFT);
        m_Parent.add(IDC_STATIC1);
        m_Layout.setShape(IDC_STATIC1, 371, 21, 55, 12);

        IDC_STATIC2 = new Label ("Relevant (Default) QCs", Label.LEFT);
        m_Parent.add(IDC_STATIC2);
        m_Layout.setShape(IDC_STATIC2, 439, 21, 77, 11);

        Clear = new Button ("Clear");
        m_Parent.add(Clear);
        m_Layout.setShape(Clear, 14, 106, 83, 17);

        ShowLogicalForm = new Button ("Scope Neutral LF");
        m_Parent.add(ShowLogicalForm);
        m_Layout.setShape(ShowLogicalForm, 13, 62, 83, 14);

        m_fInitialized = true;
        return true;
    }
}
```

VITA AUCTORIS

Sudhakar Reddy Pareddy was born in Sindhanur, India, on May 06, 1974. He came as a student to Canada in 1999. In 1998 Sudhakar received his Bachelor of Engineering (B. E) in Computer Science from Bapuji Institute of Engineering and Technology, Davanagere, Karnataka, India. Sudhakar is in the process of completing his Masters in Computer Science at the University of Windsor, and plans to pursue a Ph.D in Computer Science. His current research interest is Natural Language Understanding.